UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Extensión a la herramienta didáctica Mileva

Informe de Proyecto de Grado presentado por

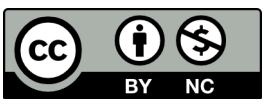
Federico Galiano, Damián Madeira

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Federico Gómez
Sylvia da Rosa

Montevideo, 4 de junio de 2026



Resumen

El presente proyecto de grado describe el diseño, la implementación y pruebas de evaluación de una extensión de la herramienta didáctica Mileva, orientada a fomentar el desarrollo del pensamiento computacional en estudiantes de educación media sin experiencia previa en programación. Mientras que la primera versión de Mileva trabajaba la sentencia de asignación, este nuevo módulo se focaliza en la *evaluación de expresiones booleanas*, abordando los operadores lógicos AND (conjunción), OR (disyunción) y NOT (negación), los comparadores relacionales y el mecanismo de evaluación por circuito corto.

El diseño de los ejercicios se sustenta en tres pilares teóricos: el modelo Hybrid Interaction System (sistema de interacción híbrida) de Schulte y Budde, el concepto pedagógico de Bildung como proceso de transformación del sujeto a través de la interacción con el artefacto digital, y la Epistemologia Genética de Jean Piaget, en particular su tríada de etapas intra-inter-trans, su Ley General de la Cognición, en la versión extendida que incorpora la conceptualización de la ejecución computacional y la noción de generalización constructiva. A partir de estos fundamentos se construyeron cinco grupos de ejercicios graduados en complejidad: variables booleanas, expresiones booleanas, operadores lógicos, evaluación por circuito corto y formalización con variables abstractas. Cada ejercicio combina un pseudocódigo simple con visualizaciones interactivas (luces, puertas, robots y cajas abstractas) que ofrecen retroalimentación visual inmediata frente a cada acción del estudiante.

La herramienta se implementó como una aplicación web de página única (Single Page Application) con la biblioteca React, persistiendo el progreso del estudiante en el almacenamiento local del navegador.

La evaluación se realizó mediante dos iteraciones de pruebas individuales con quince participantes de entre 15 y 26 años sin formación previa en programación. Entre la primera y la segunda iteración se incorporaron diversas correciones y mejoras en la herramienta. Los resultados de la segunda iteración muestran mejoras consistentes en todos los grupos de ejercicios, destacándose especialmente el grupo de circuito corto, donde la proporción de estudiantes capaces de resolver los ejercicios sin ayuda de un facilitador pasó del 0 % al 36 %. Estas evidencias indican que la herramienta logra ofrecer un entorno de construcción de conocimiento informal sobre conceptos básicos de computación que sirve como base para la formalización posterior en el sistema educativo.

Palabras clave: Mileva, pensamiento computacional, expresiones booleanas, didáctica de la programación

Índice general

1. Introducción **1**

2. Marco teórico **3**
2.1. Hybrid Interaction System (HIS) y el concepto de *Bildung* 3
2.2. Teoría de Piaget sobre construcción de conocimiento 4
2.3. Ley de la Cognición extendida 5
2.4. Generalización inductiva y constructiva 6

3. Descripción y análisis de ejercicios **7**
3.1. Variables booleanas (Grupo 1) 8
3.1.1. Ejercicio 1 9
3.1.2. Ejercicio 2 10
3.1.3. Ejercicio 3 11
3.1.4. Ejercicio 4 12
3.1.5. Ejercicio 5 13
3.1.6. Ejercicio 6 14
3.1.7. Ejercicio 7 15
3.2. Expresiones booleanas (Grupo 2) 15
3.2.1. Ejercicio 1 17
3.2.2. Ejercicio 2 18
3.2.3. Ejercicio 3 19
3.2.4. Ejercicio 4 20
3.2.5. Ejercicio 5 21
3.2.6. Ejercicio 6 22
3.2.7. Ejercicio 7 23
3.3. Operadores booleanos (Grupo 3) 24
3.3.1. Ejercicio 1 26
3.3.2. Ejercicio 2 27
3.3.3. Ejercicio 3 28
3.3.4. Ejercicio 4 29
3.3.5. Ejercicio 5 29
3.3.6. Ejercicio 6 31
3.3.7. Ejercicio 7 31
3.3.8. Ejercicio 8 32

3.3.9. Ejercicio 9 . . . 33
3.3.10. Ejercicio 10 . . . 34
3.3.11. Ejercicio 11 . . . 35
3.4. Evaluación por circuito corto (Grupo 4): . . . 36
3.4.1. Ejercicio 1 . . . 37
3.4.2. Ejercicio 2 . . . 38
3.4.3. Ejercicio 3 . . . 40
3.4.4. Ejercicio 4 . . . 42
3.4.5. Ejercicio 5 . . . 46
3.4.6. Ejercicio 6 . . . 47
3.5. Formalización (Grupo 5) . . . 50
3.5.1. Ejercicio 1 . . . 52
3.5.2. Ejercicio 2 . . . 53
3.5.3. Ejercicio 3 . . . 54
3.5.4. Ejercicio 4 . . . 55
3.5.5. Ejercicio 5 . . . 56
3.5.6. Ejercicio 6 . . . 57
3.5.7. Ejercicio 7 . . . 58

4. Arquitectura, diseño e implementación 59
4.1. Arquitectura existente y enfoque de la extensión . . . 59
4.1.1. Resumen de la arquitectura preexistente . . . 59
4.1.2. Pilares sobre los que se construyó la extensión . . . 60
4.1.3. Criterios para la conservación de la arquitectura original . 61
4.1.4. Migración a Vite como Build Tool . . . 62
4.2. Diseño de los ejercicios . . . 62
4.2.1. Visualización . . . 62
4.2.2. Lenguaje de las sentencias (extensiones para expresiones booleanas) . . . 63
4.2.3. Componentes . . . 64
4.2.4. Parser y validación de expresiones booleanas . . . 65
4.3. Implementación . . . 66
4.3.1. Ejercicios en formato JSON (extensión) . . . 66
4.3.2. Página de inicio y navegación (extensión) . . . 66
4.3.3. Tutorial y uso de la herramienta . . . 67

5. Pruebas 68
5.1. Enfoque del estudio . . . 69
5.2. Participantes . . . 69
5.3. Metodología . . . 70
5.4. Resultados . . . 70
5.4.1. Grupo 1 — Variables booleanas . . . 70
5.4.2. Grupo 2 — Expresiones booleanas . . . 71
5.4.3. Grupo 3 — Operadores lógicos (AND, OR, NOT) . . . 71
5.4.4. Grupo 4 — Evaluación de expresiones (circuito corto) . . . 73
5.4.5. Grupo 5 — Formalización . . . 74

5.5. Cambios implementados 74
5.6. Síntesis de los resultados 76

6. Conclusiones **78**
6.1. Conclusiones del proyecto 78
6.2. Trabajo futuro 79

Referencias **81**

A. Ejercicios adicionales **83**

B. Arquitectura **91**
B.0.1. Lenguaje de sentencias de la versión original 91
B.0.2. Diagrama de despliegue 92
B.0.3. Diagrama de componentes: versión original (2023) 93
B.0.4. Diagrama de componentes: versión actualizada (2026) . . 94
B.0.5. Migración a Vite y optimización de la infraestructura . . 97

Capítulo 1

Introducción

Una de las diferencias más significativas entre la comunicación humana y la comunicación humano-computadora reside en la naturaleza del lenguaje: el lenguaje natural es intrínsecamente ambiguo y dependiente del contexto, mientras que los lenguajes de programación poseen una sintaxis y una semántica determinadas que limitan lo expresado a una única interpretación. La interacción del pensamiento humano con un sistema informático a través de un lenguaje formal permite transformar el razonamiento algorítmico en **pensamiento computacional**, mediante el cual el sujeto será capaz de instruir a una máquina para resolver problemas (Salanci, 2015; Cazzola y Olivares, 2015; Grover, Jackiw, y Lundh, 2019). El presente proyecto de grado presenta una extensión denominada **Módulo 2**, de la herramienta didáctica Mileva (Rosselli, Correa, y Guayta, 2023), diseñada para fomentar esta transformación del pensamiento mediante la experimentación directa con un pseudocódigo y guiando al estudiante en la resolución de problemas sencillos, agrupados en cinco grupos de ejercicios.

Uno de los objetivos del proyecto al diseñar Mileva es conectar el diseño de los ejercicios tanto con los conceptos teóricos fundamentales (descritos en el capítulo 2) como con el conocimiento sobre los conceptos específicos de computación, desde una perspectiva didáctica. En consecuencia, el foco de Mileva es fomentar la construcción de conocimiento informal por parte del estudiante sobre conceptos básicos de computación cuya formalización se aborda en el sistema educativo. En el último grupo de ejercicios se introducen nombres simbólicos a las variables como un primer paso hacia la formalización.

Mientras que la primera etapa de Mileva (Rosselli y cols., 2023) se centró en la sentencia de asignación, el contenido específico de la extensión desarrollada en el presente proyecto se focaliza en la **evaluación de expresiones booleanas**. A través de una secuencia de actividades incrementales, el estudiante interactúa con conceptos de lógica (operadores AND, OR y NOT), comparadores relacionales ($>$, $<$, $=$, etc.) y mecanismos de optimización de ejecución como la evaluación por circuito corto.

La herramienta se basa en un sistema de interacciones donde las acciones del estudiante producen respuestas visuales inmediatas (representado por luces,

puertas o robots), permitiéndole identificar errores y autocorregirse sin necesidad de intervención docente constante.

El resto del informe se organiza de la siguiente manera:

- **Capítulo 2 — Marco teórico:** Expone los fundamentos conceptuales que orientan el diseño, integrando el modelo *Hybrid Interaction System* (HIS), el concepto de *Bildung* y la Epistemología Genética de Piaget para fundamentar el diseño de cada ejercicio.
- **Capítulo 3 — Descripción de los grupos de ejercicios:** Detalla de forma narrativa la estructura y objetivos didácticos de los cinco nuevos grupos de actividades que componen la extensión para definición y evaluación de expresiones booleanas.
- **Capítulo 4 — Arquitectura e implementación:** Describe la arquitectura, el diseño y consideraciones de implementación de la herramienta, las extensiones realizadas en el *parser* para soportar el manejo de expresiones booleanas y la organización de los ejercicios en formato JSON.
- **Capítulo 5 — Pruebas:** Presenta el análisis de las instancias de validación realizadas con estudiantes, incluyendo el registro de errores, el desempeño de los estudiantes y las mejoras implementadas a partir de los resultados obtenidos.
- **Capítulo 6 — Conclusiones:** Evalúa los resultados alcanzados frente a los objetivos propuestos y plantea líneas de trabajo futuro para la expansión de la herramienta.

Capítulo 2

Marco teórico

El presente capítulo describe brevemente los fundamentos teóricos que orientan el diseño de la herramienta didáctica y la secuencia de actividades propuestas. Se adopta la misma perspectiva del proyecto anterior (Rosselli y cols., 2023), donde la programación trasciende la visión en la cual es entendida únicamente como dominio técnico o sintáctico, para situarla en un proceso de construcción de conocimiento mediado por la interacción con artefactos digitales. En este marco, aprender a programar implica articular la acción del estudiante con la respuesta del sistema, de manera que el sujeto no solo escribe instrucciones que luego son ejecutadas, sino que interpreta resultados y corrige errores, lo que produce una reestructura progresiva de sus esquemas de pensamiento hacia un pensamiento computacional. Este enfoque se basa en el modelo Hybrid Interaction System (HIS) —en diálogo con el concepto formativo de *Bildung* (Schulte y Budde, 2018)— y la ley general de la cognición tomada de la Epistemología Genética de Jean Piaget, extendida para abarcar la construcción de conocimiento sobre programas y la descripción de Piaget sobre la generalización constructiva (da Rosa y Gómez, 2020; da Rosa Zipitría y Gómez Frois, 2019).

2.1. Hybrid Interaction System (HIS) y el concepto de *Bildung*

Para comprender el aprendizaje de la programación resulta insuficiente analizar por separado el comportamiento del estudiante o el funcionamiento del programa; el foco debe situarse en la interacción que se establece entre ambos durante la ejecución. En esta línea, el Hybrid Interaction System (HIS) (Schulte y Budde, 2018) concibe el aprendizaje como un fenómeno emergente de un acoplamiento recíproco entre un sujeto y un artefacto digital, mediado por acciones, procesos computacionales y respuestas observables.

Este enfoque se vincula con el concepto pedagógico de Bildung, que entiende la formación no como acumulación de información, sino como un proceso de transformación del modo en que el individuo se comprende a sí mismo y com-

prende el mundo. Desde esta premisa, la educación adquiere un carácter relacional: al actuar sobre un objeto cultural —en este caso, un sistema computacional— la respuesta del sistema reconfigura la interpretación que el estudiante tiene de su propia acción y de la lógica que la sustenta, lo que transforma el estado del pensamiento del estudiante.

En términos sistémicos, el HIS describe la interacción entre dos tipos de sistemas: por un lado, el ser humano como sistema autopoiético, capaz de autoorganizarse y reorganizar sus estructuras internas; por otro, la computadora como sistema alopoiético, cuyo funcionamiento es determinado por reglas externas y procedimientos formales. La relevancia didáctica de este modelo reside en que el aprendizaje no se limita a "usar" el software, sino que se produce en el ciclo de retroalimentación (acción del estudiante → procesamiento del sistema → respuesta visual o funcional → nueva acción del estudiante). Dicho ciclo permite que el estudiante confronte sus expectativas con resultados verificables, promoviendo la formulación de preguntas explicativas, (por ejemplo, "¿qué efecto produjo esta instrucción?", "¿qué condición no se cumplió?") favoreciendo el paso desde una participación instrumental hacia una actuación más autónoma y reflexiva en entornos digitales.

2.2. Teoría de Piaget sobre construcción de conocimiento

Piaget describe tres etapas para explicar la construcción de conocimiento científico, tanto a nivel de los individuos como en el desarrollo histórico de las ciencias, que describimos brevemente para el dominio de conocimiento correspondiente a la disciplina programación:

1. Etapa Intra: el sujeto se orienta a elementos aislados y a resultados inmediatos. En programación, esto se manifiesta cuando el estudiante manipula instrucciones o variables de forma puntual buscando "que funcione", sin tener conciencia detallada de cómo lo logra.

2. Etapa Inter: mediante la reflexión, se produce la toma de conciencia de la coordinación de acciones, relaciones entre objetos y transformaciones sobre ellos derivadas de la acción. Se avanza hacia la construcción conocimiento conceptual sobre operaciones (por ejemplo, condiciones, operadores lógicos, dependencias entre variables) y el estudiante puede anticipar parcialmente los efectos de las mismas.

3. Etapa Trans: se configuran sistemas de conjunto, es decir, estructuras generales, donde los casos particulares pasan a ser elementos de las mismas. En el campo de la programación, esta etapa se evidencia cuando el estudiante reconoce problemas algorítmicos que involucran una clase de soluciones (por ejemplo, los problemas de ordenación, y los distintos algoritmos a emplear según las características de los datos, así como el estudio

de los distintos algoritmos según criterios establecidos (por ejemplo el de la eficiencia)).

Piaget estudió a fondo la evolución del conocimiento a través de las etapas y formuló la Ley General de la Cognición (da Rosa y Gómez, 2020), que describe con el siguiente esquema: .

$$\mathbf{C} \leftarrow \mathbf{P} \rightarrow \mathbf{C'}$$

donde P representa la *periferia*, entendida como la reacción inmediata orientada al resultado; C refiere a la toma de conciencia de la coordinación de acciones y C′ refiere a la toma de conciencia de las propiedades de los objetos y las transformaciones producidas en ellos por las acciones.

2.3. Ley de la Cognición extendida

La ley de Piaget describe la construcción de conocimiento sobre algoritmos y estructuras de datos pero resulta insuficiente para abarcar la construcción de conocimiento sobre programas donde entra en juego la representación computacional de los problemas y soluciones y el análisis de la ejecución.

Para abarcar estos aspectos, se extiende la ley de Piaget (da Rosa, 2018; da Rosa y Gómez, 2020) que agrega un renglón al diagrama de Piaget, donde newP representa la situación en la cual ya se ha construido una solución algorítmica al problema y los nuevos centros newC y newC' representan la conceptualización de la ejecución de las instrucciones por parte del programa (representaciones computacionales de los datos y operaciones del algoritmo y de las modificaciones de las estructuras de datos producidas en la ejecución, como ser, cambios de estado en memoria, actualización de variables, efectos colaterales) respectivamente.

$$\begin{array}{ccccc} & & \underbrace{C \leftarrow P \rightarrow C'} & & \\ newC & \longleftarrow & newP & \longrightarrow & newC' \end{array}$$

Notar que la ley extendida comprende los dos renglones, el primero que representa la conceptualización del algoritmo, el segundo la conceptualización

de los aspectos computacionales, ambos unidos por la llave que representa la construcción de conocimiento sobre programas, como texto y como objeto ejecutable.

2.4. Generalización inductiva y constructiva

Piaget habla de generalización inductiva cuando una persona que ha resuelto exitosamente un problema, intenta aplicar el conocimiento ya construido para solucionar otros similares, sin tener en cuenta las diferencias que exigen nuevas construcciones. Piaget llama generalización constructiva a la herramienta por la cual una persona adapta y/o modifica el conocimiento construido para considerar similitudes y diferencias de un nuevo problema a resolver. La estrategia didáctica para facilitar el proceso de adaptación del conocimiento a las nuevas condiciones consiste en presentar al estudiante varios problemas con similitudes y diferencias, de modo de que tome conciencia de los aspectos críticos involucrados (por ejemplo, en el primer grupo de ejercicios, se plantea un problema similar variando el objeto (lámparas o puertas), para ayudar al estudiante a discernir que lo importante es el estado de la variable que representa al objeto.

Los principios teóricos descritos aquí fundamentan las decisiones tomadas en el diseño de los ejercicios de Mileva especialmente para la etapa de construcción de conocimiento informal sobre conceptos básicos de computación (da Rosa, Cabezas, Viera, y Gómez, 2023; Gómez y da Rosa, 2022).

Capítulo 3

Descripción y análisis de ejercicios

Al igual que en la primera versión de la herramienta Mileva (Rosselli y cols., 2023), la extensión que implementamos ofrece la posibilidad de que el estudiante enfrente y resuelva una variedad de ejercicios en este caso referidos a la temática de booleanos, en los cuales ponen en práctica conceptos construyendo conocimiento informal, antes de las etapas de formalización propias del sistema educativo. Cada ejercicio propone una secuencia de interacción entre la acción del estudiante y la respuesta del sistema, utilizando situaciones conocidas por el estudiante y un pseudocódigo sencillo poniendo el foco en la comprensión de los enunciados y la exploración y verificación de resultados. En el caso de este proyecto el concepto abordado es el de definición y evaluación de expresiones booleanas, escalonado en los siguientes grupos de ejercicios: 3.1, variables booleanas, 3.2, expresiones booleanas. 3.3, operadores booleanos, 3.4, evaluación por circuito corto, 3.5, ejercicios de formalización.

En todos los grupos, la secuencia de ejercicios busca consolidar mediante la repetición de acciones y variación de objetos la toma de conciencia del estado inicial de las variables que representan esos objetos (encendido/apagado (bombitas) o abierta/cerrada (puertas), etc) y las modificaciones de estos estados mediante las acciones representadas por las instrucciones escritas en MILEVA. Esta estrategia didáctica se basa en los principios de interacción con este tipo de herramienta descritos en Bildung (ver capítulo. 2 Marco teórico), que explican que para alcanzar un objetivo específico es necesaria cierta comprensión del enunciado y del estado inicial de las variables así como cierta reflexión sobre el resultado a alcanzar y las acciones que deben modificar dicho estado para lograrlo.

3.1. Variables booleanas (Grupo 1)

El concepto de variable constituye uno de los primeros puntos de encuentro entre el pensamiento matemático y el pensamiento computacional. En la enseñanza de la matemática, las variables suelen presentarse como símbolos que representan valores dentro de una expresión o ecuación. Sin embargo, en programación el concepto adquiere una dimensión distinta: una variable representa una posición en memoria cuyo contenido puede cambiar durante la ejecución de un programa. Esta diferencia conceptual es particularmente relevante para los estudiantes que se aproximan por primera vez a la programación, ya que tienden a interpretar el símbolo "=" como una igualdad matemática y no como una operación de asignación que modifica el estado del sistema (Rosselli y cols., 2023).

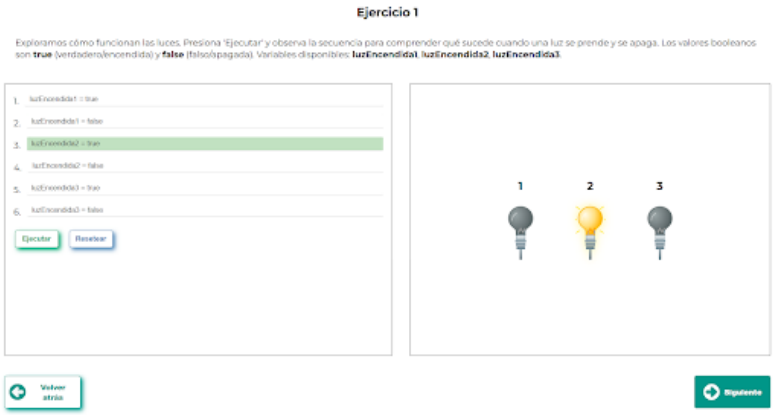
En este bloque inicial de diez ejercicios (los cuales algunos se pasaron al anexo) se introduce el uso de variables booleanas, es decir, variables que solo pueden adoptar dos valores posibles: verdadero (true) o falso (false). A través de situaciones visuales simples —como luces que se encienden o puertas que se abren— los ejercicios permiten que el estudiante *observe* cómo la ejecución de acciones sobre valores lógicos almacenados en memoria modifican propiedades de objetos dentro del entorno digital. Se busca que el estudiante establezca una primera relación entre el texto del programa y las transformaciones que ocurren en el sistema.

Los seis primeros ejercicios buscan que el estudiante se familiarice con el tipo de variables e instrucciones, planteando tareas sencillas. Los ejercicios que fueron enviados al anexo apuntan a reforzar la comprensión de la naturaleza secuencial de la ejecución computacional y cómo las instrucciones individuales afectan el estado del sistema en cada paso intermedio. En dichos ejercicios se plantea una asignación secuencial al introducir una tercera variable y requerir la gestión simultánea de los estados de las tres variables. En el ejercicio 6 de este grupo el estudiante debe considerar una restricción lógica externa (una alarma) al diseñar la secuencia de instrucciones para lograr el objetivo y evitar un estado prohibido (el conflicto lógico).

En el ejercicio 7 el estudiante se enfrenta al desafío de comprender la noción de incertidumbre sobre el estado inicial, induciéndolo a pasar de una programación basada en la observación a una programación basada en la inicialización y la anticipación del resultado, (por ejemplo, estando el estado inicial oculto, el estudiante intuitivamente debería pensar de forma similar a "al inicio una de las puertas debe estar cerrada independiente de la otra y . . . ")

A continuación se describen detalladamente los del grupo 1 que terminaron formando parte de esta extensión de Mileva y se incluyen en el anexo el resto de ejercicios que diseñamos pero tomamos la decisión de no incluir en la version final de la herramienta (en el capítulo 5.5 se explicará el por qué de esta decisión).

3.1.1. Ejercicio 1



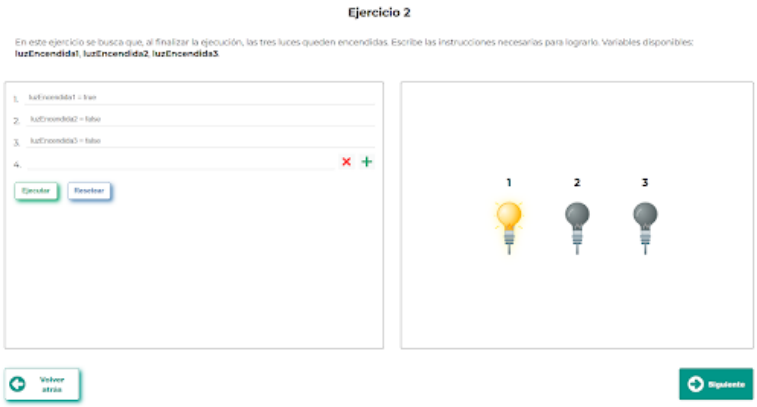
El ejercicio busca que el estudiante se concentre en las propiedades de objetos aislados (la bombita 1, la 2 o la 3) y en cómo una acción directa (la ejecución de la instrucción) modifica dichas propiedades, como un primer paso hacia la conceptualización de la sentencia de asignación de variables booleanas, según la ley de la toma de conciencia, descrita en el Capítulo 2.

• **Enunciado y Escenario:** El sistema presenta tres bombitas apagadas y una secuencia de seis instrucciones pre-cargadas. La consigna invita al usuario a presionar "Ejecutar" para comprender qué sucede cuando una luz se prende y se apaga.

Al ejecutar, Mileva resalta en verde la instrucción activa (por ejemplo, luzEncendida2 = true). El estudiante observa el efecto inmediato en el panel derecho: la bombita correspondiente cambia de un tono gris a un amarillo brillante, representando el valor true.

Este ejercicio funciona como una toma de contacto inicial con el concepto de estado booleano. Se presenta como una actividad puramente exploratoria y demostrativa donde el estudiante observa cómo el autómata procesa valores lógicos para alterar la realidad de objetos digitales simples.

3.1.2. Ejercicio 2



Se busca consolidar mediante la repetición el tránsito hacia la toma de conciencia del estado de las variables (encendido/apagado) y las modificaciones del mismo mediante la acción, que en este caso debe ser agregada por el estudiante.

• **Enunciado y Desafío:** El sistema presenta tres bombitas, donde inicialmente solo la bombita 1 está encendida. La consigna es: *"Lograr que, al finalizar la ejecución, las tres luces queden encendidas"*.

• **Estado Inicial del Código:** El panel de código muestra tres instrucciones pre-cargadas:

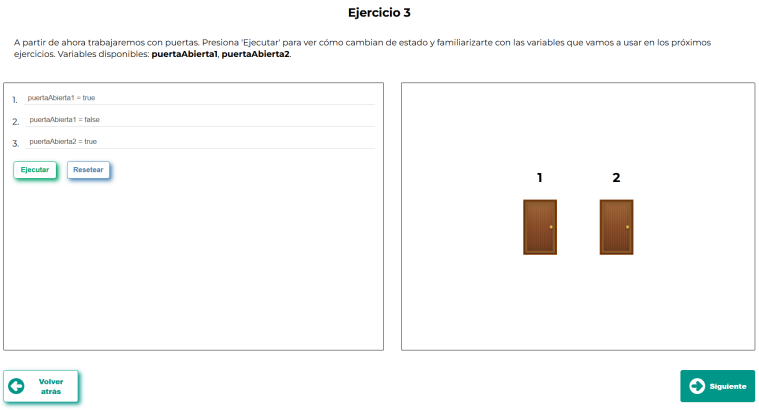
1. luzEncendida1 = true
2. luzEncendida2 = false
3. luzEncendida3 = false

• **Resolución:** El estudiante debe agregar nuevas líneas de código (haciendo clic en el botón +) para asignar el valor true a las variables luzEncendida2 y luzEncendida3.

Observar que si el estudiante simplemente presiona "Ejecutar" sin añadir nuevas instrucciones, verá cómo las bombitas 2 y 3 permanecen apagadas. Esta visualización del fracaso induce al estudiantes a razonar que para encender la luz, el valor asignado debe ser necesariamente el opuesto al actual (false → true) (el sistema no permite realizar cambios en las instrucciones iniciales, sino que es necesario añadir nuevas asignaciones).

Este ejercicio es fundamental porque, aunque parece simple, obliga al estudiante a comprobar y a reflexionar que el computador ejecuta las instrucciones de forma secuencial. Si el estudiante agrega luzEncendida2 = true en la línea 4, verá el cambio de estado sólo después de que se procesen las líneas anteriores.

3.1.3. Ejercicio 3



Este ejercicio introduce un cambio en el contexto visual de la herramienta **Mileva**, pasando de bombitas a puertas, para consolidar la comprensión del tipo de dato booleano a través de la variación, como se explicó en el punto 2.4 sobre generalización.

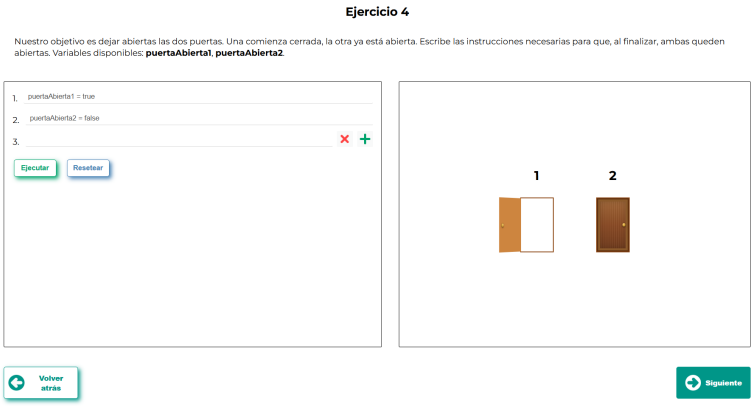
• **Enunciado y Desafío:** El sistema presenta dos puertas cerradas. El texto instruye al usuario a presionar "Ejecutar" para observar cómo cambian de estado, permitiéndole conocer las variables que se utilizarán en retos posteriores.

• **Estado del Código:** El panel de instrucciones contiene tres sentencias pre-cargadas que se ejecutan de forma secuencial:

1. puertaAbierta1 = true (Abre la puerta 1).
2. puertaAbierta1 = false (Cierra la puerta 1).
3. puertaAbierta2 = true (Abre la puerta 2).

Al ser un ejercicio de observación inicial, el estudiante actúa como un observador del ciclo de interacción: la computadora procesa el código, resalta la línea activa en verde y visualiza el efecto físico inmediato en el panel derecho.

3.1.4. Ejercicio 4



Al igual que en el ejercicio 2, en el ejercicio 4 se busca consolidar mediante la repetición el tránsito hacia la toma de conciencia del estado de las variables y las modificaciones del mismo mediante la acción. También en este caso el estudiante debe intervenir activamente, agregando las instrucciones necesarias para alcanzar el objetivo, experimentando que las interacciones están gobernadas por reglas que él puede dominar y transformar.

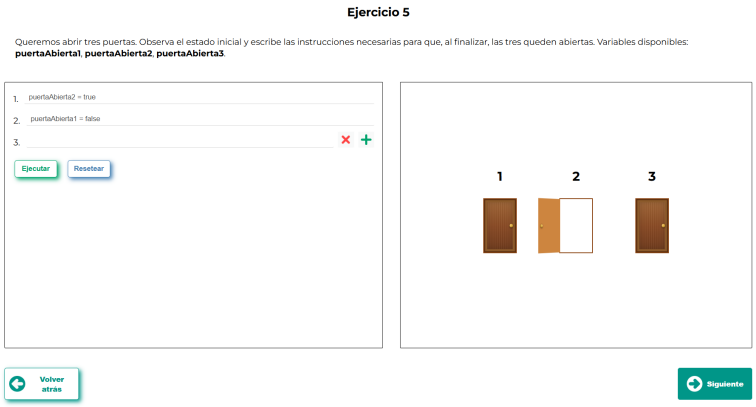
• **Enunciado y Desafío:** El sistema presenta dos puertas. La consigna indica: *"Nuestro objetivo es dejar abiertas las dos puertas. Una comienza cerrada, la otra ya está abierta. Escribe las instrucciones necesarias para que, al finalizar, ambas queden abiertas"*.

• **Estado Inicial del Código:** El panel de instrucciones muestra:

1. puertaAbierta1 = true (Refleja que la puerta 1 ya está abierta).
2. puertaAbierta2 = false (Refleja que la puerta 2 está cerrada).

• **Resolución:** El estudiante debe interactuar con el panel de código agregando una nueva línea para asignar el valor true a la variable puertaAbierta2.

3.1.5. Ejercicio 5



Este ejercicio eleva el nivel de complejidad al introducir una tercera variable y requerir la gestión simultánea de múltiples estados. Representa un paso avanzado en la construcción del concepto de asignación secuencial.

• **Enunciado y Desafío:** El sistema presenta tres puertas. La consigna indica: *"Queremos abrir tres puertas. Observa el estado inicial y escribe las instrucciones necesarias para que, al finalizar, las tres queden abiertas"*.

• **Estado Inicial del Escenario:**

◦ La puerta 2 se visualiza abierta.

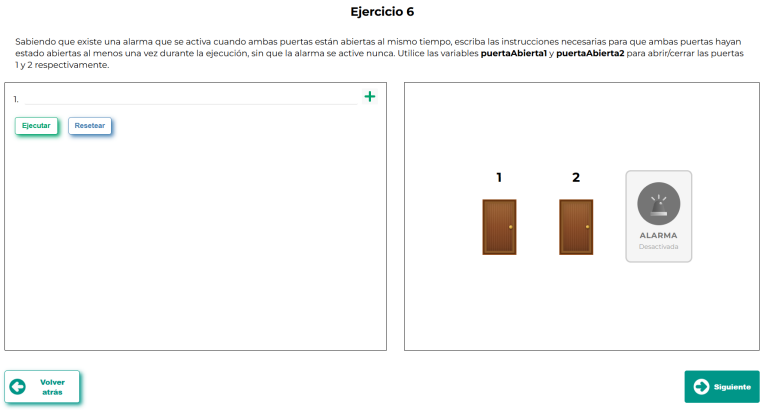
◦ Las puertas 1 y 3 se visualizan cerradas.

• **Estado Inicial del Código:** El panel muestra dos sentencias pre-cargadas que reflejan parte de la realidad visual, pero que contienen un "conflicto" intencional para inducir a la reflexión:

1. puertaAbierta2 = true (Mantiene la puerta 2 abierta).
2. puertaAbierta1 = false (Mantiene la puerta 1 cerrada).

• **Resolución:** El estudiante debe realizar una intervención múltiple: utilizar el botón + para agregar una nueva instrucción para asignar el valor true a puertaAbierta2 y utilizar nuevamente el botón + para agregar otra nueva línea de código que asigne el valor true a la variable puertaAbierta3. Este es el primer ejercicio en el que el estudiante debe agregar dos instrucciones para cumplir con la consigna planteada.

3.1.6. Ejercicio 6



Este ejercicio introduce una restricción lógica externa (la alarma), lo que eleva el desafío desde una simple asignación de estados hacia la gestión de procesos secuenciales y la comprensión del estado global del sistema.

• **Enunciado y Desafío:** El escenario presenta dos puertas cerradas y una alarma desactivada. La consigna indica: *"Sabiendo que existe una alarma que se activa cuando ambas puertas están abiertas al mismo tiempo, escriba las instrucciones necesarias para que ambas puertas hayan estado abiertas al menos una vez durante la ejecución, sin que la alarma se active nunca"*.

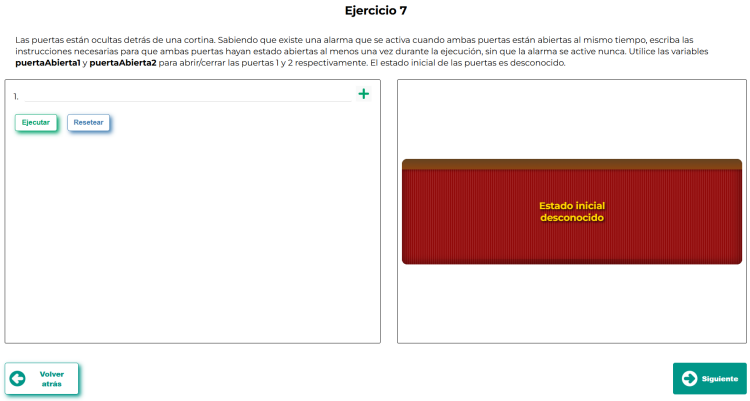
• **Estado Inicial:** El panel de código está vacío, lo que otorga al estudiante total autonomía para proponer la solución desde cero.

• **Resolución:** El estudiante debe diseñar una secuencia de instrucciones (utilizando el botón +) que alterne los estados de las puertas. Por ejemplo:

1. puertaAbierta1 = true (Abre la primera).
2. puertaAbierta1 = false (Cierra la primera antes de abrir la otra).
3. puertaAbierta2 = true (Abre la segunda).

Con este ejercicio se busca preparar al estudiante para niveles más altos de formalización, al demostrar que programar no es solo asignar valores, sino gobernar un flujo de estados que debe respetar reglas lógicas predefinidas.

3.1.7. Ejercicio 7



Con este ejercicio se busca que el sujeto comprenda que la secuencia de instrucciones debe funcionar independientemente de los valores previos que existan en la memoria (representada aquí por lo oculto tras las puertas tras el telón). Se introduce el concepto de incertidumbre sobre el estado inicial, planteando el nivel de abstracción más alto dentro de este bloque de ejercicios variables de estado. Obliga al estudiante a pasar de una programación basada en la observación a una programación basada en la inicialización y en lograr estados seguros del sistema previo a la ejecución.

• **Enunciado y Desafío:** El sistema presenta un telón rojo con el texto "Estado inicial desconocido". La consigna es idéntica al ejercicio anterior: lograr que ambas puertas hayan estado abiertas al menos una vez sin que la alarma se active nunca. Sin embargo, el estudiante debe escribir sus instrucciones sin saber si las puertas comienzan abiertas o cerradas.

• **Estado Inicial:** El panel de código está vacío. El panel de visualización está bloqueado visualmente por el telón.

• **Resolución:** Para garantizar el éxito sin importar el estado inicial, el estudiante debe adoptar una estrategia de "limpieza" o inicialización. Una solución experta consistiría en:

1. puertaAbierta1 = false (Asegura que la puerta 1 esté cerrada).
2. puertaAbierta2 = false (Asegura que la puerta 2 esté cerrada, desactivando cualquier riesgo de alarma inicial).
3. Proceder con la secuencia de apertura alternada (ej. abrir 1, cerrar 1, abrir 2).

3.2. Expresiones booleanas (Grupo 2)

Los ejercicios de este grupo del 1 al 6 incluido, plantean el pasaje de la asignación simple de valores (true/false) a la asignación de expresiones lógicas.

Aquí, el valor de una variable booleana ya no se decide de forma directa por el usuario, sino que es el resultado de la evaluación de una expresión booleana realizada por el autómata. En este grupo las expresiones booleanas se construyen con operadores relacionales ($\geq$, $\leq$, $<$, $>$, $==$, $\neq$, etc). Las expresiones usando operadores booleanos AND, OR y NOT se clasifican en el próximo grupo de ejercicios.

La estrategia didáctica es la descrita al comienzo de este capítulo: los ejercicios presentan pequeñas variaciones buscando que el estudiante experimente diversas situaciones que involucran distintas comparaciones entre valores de las variables. Por ejemplo, en el ejercicio 6 se induce el razonamiento intuitivo sobre la negación al plantear un objetivo de negación lógica por medio de los datos. A diferencia de los retos planteados en los ejercicios anteriores donde se buscaba que una condición fuera verdadera, aquí el estudiante debe manipular el estado de una variable de entrada para forzar un resultado **falso** en la expresión final.

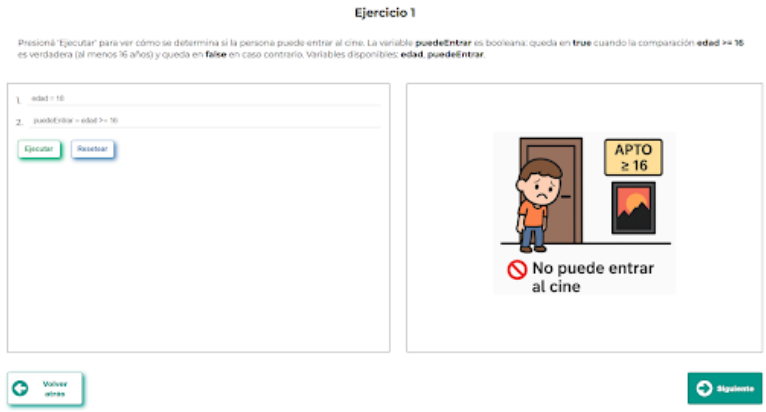
Asimismo, al utilizar nombres de variables como *temperatura* y *haceFrio*, el estudiante experimenta con las variables como celdas de memoria con roles específicos (una como entrada de sensor y otra como bandera de estado), construyendo conocimiento *informal* sobre un concepto computacional básico.

Al mismo tiempo, se induce la comprensión intuitiva del estudiante sobre la semántica de los operadores relacionales (como $\geq$) como herramientas para establecer umbrales de decisión. Por ejemplo, en el ejercicio 5 observa que el valor de la variable *haceCalor* no es una asignación directa, sino el producto de una condición que el autómata evalúa en tiempo de ejecución.

El bloque culmina con el ejercicio 7 que utiliza un teclado de alarma para obligar al estudiante a distinguir entre **definir** una variable (uso de $=$) y **preguntar** sobre ella (uso de $==$). Mediante este ejercicio se busca reforzar la necesidad de superar la ambigüedad de la matemática sobre el $=$, preparando al estudiante para el aprendizaje de estructuras de control más complejas, como la selección condicional, donde la toma de decisiones del autómata dependerá siempre de una evaluación de igualdad o comparación previa.

A continuación se describen detalladamente los ejercicios del grupo 2 que terminaron formando parte de esta extensión de Mileva y se incluyeron en el anexo el resto de ejercicios que diseñamos pero tomamos la decisión de no incluir en la versión final de la herramienta (en el capítulo 5.5 se explicará el por qué de esta decisión).

3.2.1. Ejercicio 1



• **Enunciado y Desafío:** El sistema presenta una situación cotidiana: el ingreso a una película apta para mayores de 16 años. El estudiante debe observar cómo la variable puedeEntrar depende del valor asignado a edad.

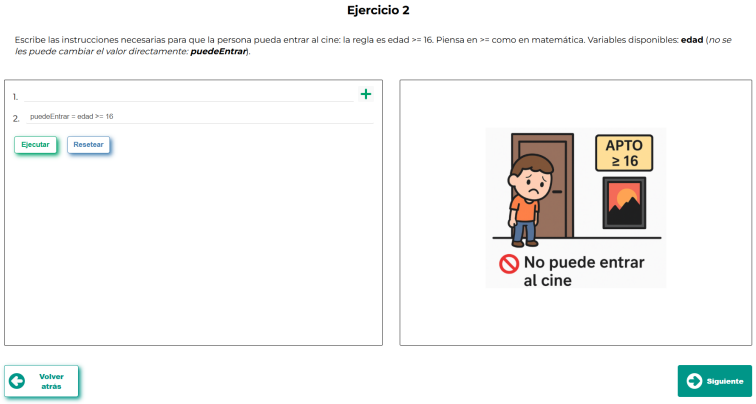
• **Resolución:**

◦ **Parte 1 (Estado de Observación):** El panel muestra una inicialización pre-cargada: edad = 18. Debajo, la expresión lógica $puedeEntrar = edad \geq 16$. En el panel visual, se ve inicialmente a un niño triste con un cartel de "No puede entrar", representando el estado *antes* de que el computador procese la lógica.

◦ **Parte 2 (Efecto de la Ejecución):** Al presionar "Ejecutar", la herramienta procesa las líneas secuencialmente. Cuando llega a la línea 2 (resaltada en verde), el computador evalúa si $18 \geq 16$. Al ser verdadero, la variable puedeEntrar cambia su estado interno a true, y el panel visual se transforma: el niño ahora sonríe y entra al cine bajo el mensaje "Puede entrar al cine". Este es un

ejercicio introductorio donde se marcan las reglas conceptuales que el estudiante utilizará en próximos ejercicios.

3.2.2. Ejercicio 2



En este ejercicio el estudiante debe intervenir activamente para definir el estado inicial del sistema, conectando con el concepto de inicialización de variables y con la comprensión de que el autómata requiere un dato de entrada guardado en memoria para poder ejecutar una comparación lógica posterior.

Esta comprensión, ligada a una relación de causalidad entre datos y lógica, es fundamental para avanzar hacia estructuras de control más complejas como la selección (if) o la iteración (las cuales se podrían considerar para una nueva versión de Mileva).

Desde la perspectiva de **Bildung**, este ejercicio busca transformar al estudiante de un usuario que "mira cómo funciona" a un programador que define las condiciones para que el sistema funcione, otorgándole control sobre la "maquinaria"lógica detrás de la pantalla.

• **Enunciado y Desafío:** Se mantiene el contexto del cine con la regla "APTO $\geq$ 16". La consigna pide escribir las instrucciones necesarias para que la persona pueda entrar. El panel de visualización muestra inicialmente al niño triste y el cartel de "No puede entrar".

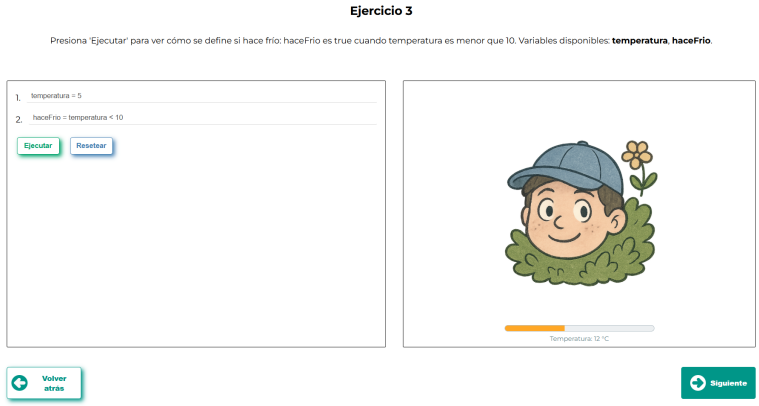
• **Resolución:**

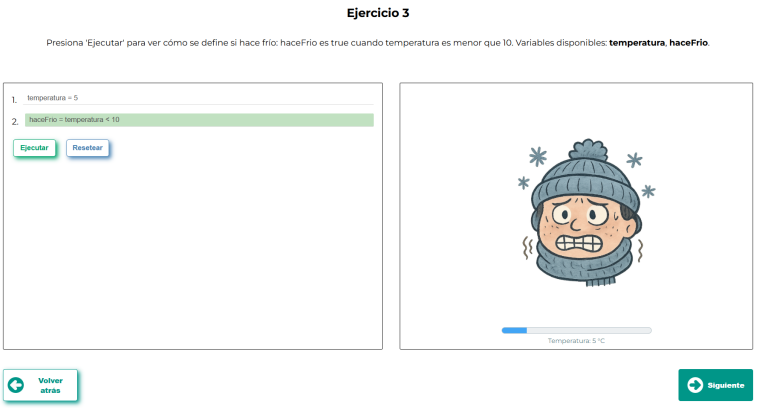
◦ El panel de código presenta la línea 1 vacía y la línea 2 con la expresión pre-cargada: puedeEntrar = edad $\geq$ 16.

◦ El estudiante debe agregar una sentencia que asigne un valor numérico a la variable edad (ej. edad = 18).

◦ Si el estudiante intenta ejecutar el código sin inicializar edad, el sistema no podrá evaluar la comparación, forzando la reflexión sobre la necesidad de contar con datos preexistentes en las celdas de memoria.

3.2.3. Ejercicio 3





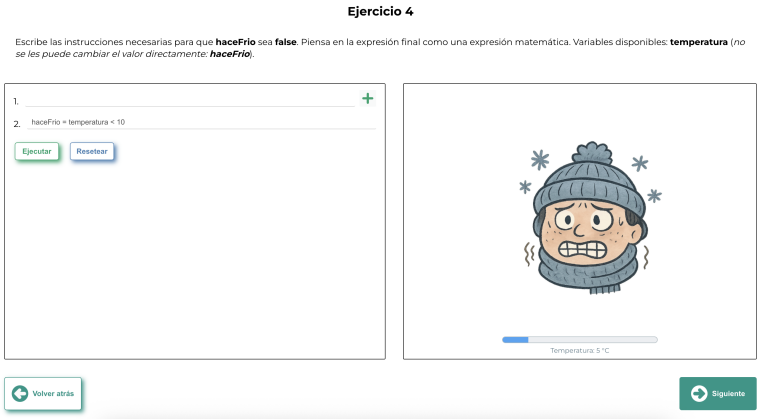
Este ejercicio introductorio profundiza en la comprensión de las expresiones relacionales al plantear un objetivo de negación lógica por medio de los datos. A diferencia de los retos anteriores donde se buscaba que una condición fuera verdadera, aquí el estudiante observa como el hecho de cambiar el estado de una variable de entrada lleva cambiar un resultado **falso** en la expresión final. Si bien es un ejercicio introductorio, el estudiante debe observar e identificar qué ciertos valores de una variable numérica invalidan una comparación específica (< 10) y transforman la realidad visual del sistema.

- **Enunciado y Desafío:** Este es un ejercicio introductorio. El sistema muestra un panel visual donde el niño está abrigado y temblando de frío (estado haceFrio = true). La consigna indica: *"Escribe las instrucciones necesarias para que haceFrio sea false"*.
- **Resolución:**

◦ El estudiante debe inicializar el ejercicio y se despliega a modo introductorio un tutorial para familiarizarse con el contexto climático en el que darán lugar esta serie de ejercicios.

◦ Al presionar "Ejecutar", la herramienta procesa el código secuencialmente. Al evaluarse la línea 2, el computador determina que la condición se cumple, el niño comienza a temblar y el entorno cambia a un clima frío.

3.2.4. Ejercicio 4



En este ejercicio, siguiendo el contexto climático, el estudiante debe comprender cómo una condición lógica (< 10) traduce un dato cuantitativo (grados centígrados) en una categoría cualitativa booleana (hace frío o no). En este ejercicio se busca poner en práctica lo visto en el ejercicio anterior, ahora se busca que el estudiante comprenda la relación de dependencia entre variables de distinto tipo: el valor de la variable booleana haceFrio no es estático, sino que depende enteramente del valor asignado a la variable numérica temperatura y de la regla de comparación establecida, por lo tanto el éxito del ejercicio depende del valor que el estudiante le asigne a la variable temperatura

• **Enunciado y Desafío:** Escribe las instrucciones necesarias para que haceFrio sea false. Piensa en la expresión final como una expresión matemática. Variables disponibles: temperatura (no se les puede cambiar el valor directamente: haceFrio).

• **Resolución:**

◦ **Parte 1 (Estado Inicial):** El panel de código contiene:

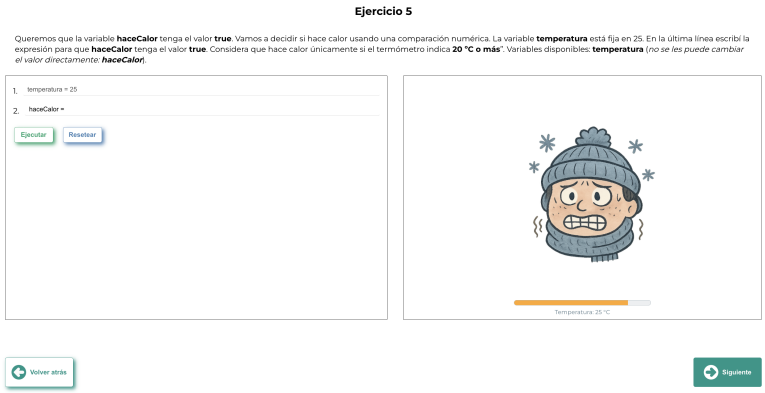
1.(Línea en blanco, donde el usuario debe poner el valor de temperatura que crea correcto)

2. haceFrio = temperatura < 10

En la visualización inicial, el niño aparece cabrigado, tiembla de frío y aparecen copos de nieve.

◦ **Parte 2 (Efecto de la Ejecución):** Al presionar "Ejecutar", Mileva resalta las líneas con color verde siguiendo la secuencia. La computadora realiza la evaluación lógica ($5 < 10 = true$). Instantáneamente, la visualización cambia: Una barra de progreso inferior indica visualmente que la temperatura ha cambiado al valor asignado, y si es el valor correcto el niño aparece una gorra y una flor (clima cálido), representando el estado de la interfaz. En caso de que el estudiante falle al resolver el ejercicio (asignó un valor erróneo), existe un feedback visual donde aparece un mensaje de error el cual indica al estudiante que el valor asignado no es correcto y provee sugerencias para ayudarlo a resolver el ejercicio.

3.2.5. Ejercicio 5



Este ejercicio complementa al anterior al invertir el rol del estudiante: en lugar de inicializar el dato de entrada, ahora debe definir la sentencia final utilizando la variable temperatura cuyo valor está fijo. Se busca que el estudiante comprenda la semántica de los operadores relacionales (como $\geq$) como herramientas para establecer umbrales de decisión en sentencias que contienen variables las cuales tienen cierto valor. El estudiante debe identificar que el valor de la variable haceCalor no es una asignación directa, sino el producto de una condición que el autómata evalúa en tiempo de ejecución. Observando el valor inicial de la variable *temperatura*, el estudiante tiene la posibilidad de escribir distintas condiciones correctas (por ejemplo temperatura $\geq$ distintos valores) que asignen a la variable *haceCalor* el valor true.
Por otro lado, si el estudiante utiliza un operador incorrecto (ej. temperatura $<$ 20), Mileva visualizará el "fracaso" al mantener al niño abrigado a pesar de que hay 25 °C. Este conflicto induce al estudiante a buscar la razón del fracaso en la sintaxis de la comparación que llevó al autómata a interpretar los símbolos matemáticos, mostrando el poder de la interacción como se describe en *Bildung* (Schulte y Budde, 2018). Además del feedback visual también se muestra un mensaje de error en este caso que indica al estudiante que, por ejemplo, para

una temperatura de 20 grados, dicha condición no hace que haceCalor valga true, con lo cual no cubre todos los casos que solicita la consigna y se debe escribir una nueva expresión con una condición más abarcativa.

• **Enunciado y Desafío:** El sistema presenta una situación donde la temperatura está fija en 25 °C. El estudiante debe escribir la expresión lógica para que haceCalor sea true, considerando que el calor se define como una temperatura de "20 °C o más".

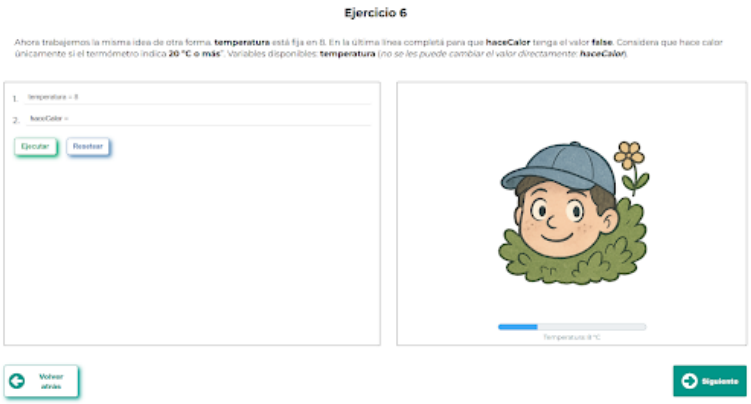
• **Estado Inicial del Código:** El panel muestra:

1. temperatura = 25 (Valor fijo de entrada).
2. haceCalor = (Espacio vacío para que el estudiante complete la expresión).

• **Resolución:** El estudiante debe completar la línea 2 escribiendo temperatura ≥ 20 o bien $20 \leq$ temperatura. Al presionar "Ejecutar", Mileva procesará la comparación. Como 25 es mayor o igual a 20, la variable cambiará a true, lo que disparará la animación de éxito donde el niño cambia su vestimenta de invierno por una de verano.

Como caso particular, si el estudiante escribiese temperatura ≥ 23 por ejemplo, el ejercicio también se completaría exitosamente puesto que haceCalor también valdría true con la temperatura inicial dada.

3.2.6. Ejercicio 6



Este ejercicio complementa al anterior al exigir que el estudiante formalice la misma regla lógica pero bajo un escenario donde el resultado debe ser negativo (false). El desafío no radica en cambiar la lógica, sino en comprender que una expresión formal es una sentencia que puede arrojar distintos resultados según los datos de entrada. Se busca que el estudiante experimente que la expresión lógica (temperatura ≥ 20) es una representación genérica de una regla del mundo físico que el autómata evalúa objetivamente. Se busca tambien desacoplar la idea de que true es positivo y false es negativo, sino que son ciertos valores que una ejecución puede arrojar. Por otro lado, este ejercicio induce a la reflexión del estudiante tratando de evitar que aprenda por ensayo y error mecánico,

ya que al forzarlo a usar una temperatura que producirá un resultado false, se pone el acento en una coordinación de acciones y decisiones rigurosas. Se busca que el estudiante deje de ser un espectador para convertirse en un actor que experimenta la naturaleza algorítmica de la evaluación de expresiones donde su accionar es influenciado por el resultado de las evaluaciones del autómata.

Siguiendo el modelo HIS, la interacción entre el humano y el artefacto digital produce una transformación mutua:

• **Acción del Humano ($H1$):** El estudiante traduce una regla cualitativa del lenguaje natural a una expresión en el formalismo de Mileva.

• **Acción de la Computadora ($C1$):** El sistema procesa la lógica y visualiza el estado. En este caso, el niño permanece en un estado neutral o fresco.

El estudiante experimenta la noción de que "falso"no significa "error", sino que es un valor de verdad tan válido y necesario como "verdadero"para representar el mundo digital.

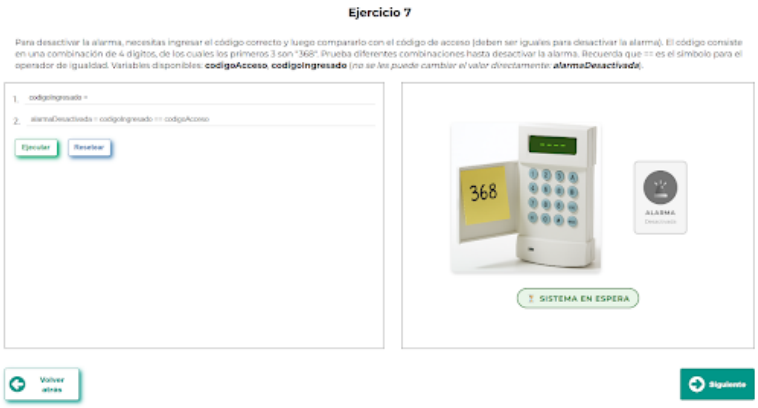
• **Enunciado y Desafío:** La temperatura está fija en 8 °C. La consigna indica: *"Completá la última línea para que haceCalor tenga el valor false. Considera que hace calor únicamente si el termómetro indica 20 °C o más".*

• **Estado Inicial del Código:** El panel presenta:

1. temperatura = 8 (Entrada de dato que no cumple el umbral de calor).
2. haceCalor = (Espacio para que el estudiante defina la regla).

• **Resolución:** El estudiante debe escribir la expresión temperatura ≥ 20 o bien $20 \geq$ temperatura. Al presionar "Ejecutar", el autómata comparará 8 con 20. Al ser el resultado false, se valida el objetivo del ejercicio.

3.2.7. Ejercicio 7



• **Enunciado y Desafío:** El sistema presenta una alarma activa y un teclado numérico. El código de acceso tiene 4 dígitos, pero solo se conocen los primeros tres ("368"). El estudiante debe probar las 10 combinaciones posibles para el último dígito hasta que la variable alarmaDesactivada evalúe a true.

A través de un contexto de seguridad (un teclado numérico), el estudiante debe interactuar con el sistema para descubrir un dato oculto mediante la validación lógica.

• **Estado Inicial del Código:** El panel muestra:

1. codigoIngresado = (Espacio para que el estudiante complete el número de 4 dígitos).

2. alarmaDesactivada = codigoIngresado == codigoAcceso (Expresión lógica pre-cargada que realiza la comparación).

Al ver las dos líneas de código juntas, el estudiante percibe visualmente que:

1. La primera línea **define** una realidad, por ejemplo (codigoIngresado = 3685).

2. La segunda línea **pregunta** sobre esa realidad (codigoIngresado == codigoAcceso).

• **Resolución:** El estudiante debe realizar una búsqueda sistemática. Escribe una instrucción (ej. codigoIngresado = 3680), presiona "Ejecutar" y observa el resultado visual. Si la alarma permanece encendida, debe "Resetear" y probar con el siguiente dígito (3681, 3682, etc.) hasta lograr la coincidencia exacta con codigoAcceso.

Este ejercicio introduce una distinción semántica fundamental en la programación: la diferencia entre la **asignación de un valor** (=) y la **comparación por igualdad** (==).

Se busca que el estudiante experimente al operador de igualdad (==) como una herramienta que permite al autómata comparar una variable con un valor en este caso numérico y devolver un valor booleano que nos dice si esa evaluación es verdadera o falsa. Esta experiencia se vincula con el concepto de que la igualdad computacional es una expresión booleana que evalúa si dos contenedores poseen el mismo contenido.

3.3. Operadores booleanos (Grupo 3)

En este grupo de ejercicios se introducen expresiones booleanas empleando los operadores **AND, OR y NOT**. Diversos trabajos en didáctica han abordado la enseñanza de la lógica booleana mediante enfoques lúdicos y software gamificado (Weng, Tseng, y Lee, 2010; Jiménez-Hernández, Oktaba, Díaz-Barriga, y Piattini, 2020; Sabitzer, Pasterk, y Reçi, 2014). De acuerdo a nuestro marco teórico, en esta extension de Mileva y en particular en este grupo de ejercicios optamos por la visualizacion de un asistente robótico al cual le haremos realizar ciertas tareas, y mientras el estudiante enfrenta diversos desafíos experimentará situaciones que involucran a cada uno de los conceptos.

Los ejercicios 1, 2, 4 y 6 cumplen un rol de tutorial, proponiendo actividades puramente exploratorias e ilustrativas para introducir al estudiante en las variables que se usarán en este grupo.

El ejercicio 1 constituye la toma de contacto inicial con un nuevo entorno visual y funcional en **Mileva**: el robot barrendero. Al igual que en el inicio del Grupo 1 con los ejercicios de luces y puertas, este ejercicio se presenta como

una actividad puramente exploratoria y demostrativa para que el estudiante comprenda el comportamiento de las variables en cada uno de los estados de este módulo.

El ejercicio 2 introduce un nuevo desafío: el paso de la asignación simple de valores a la asignación de expresiones compuestas. Al igual que los inicios de otros grupos, este ejercicio se presenta como un tutorial con instrucciones fijas para que el estudiante actúe como un observador del proceso lógico del autómata.

Los ejercicios 3 y 5 representan el paso de la observación tutorial a la intervención activa del estudiante en el entorno del robot barrendero.

En los ejercicios 6 y 7 se introduce al estudiante en la semántica del operador NOT (negación) mediante un formato tutorial de observación guiada. Se busca inducir al estudiante a comprender la semántica del operador NOT como una herramienta de inversión de estado, donde la acción de la computadora es procesar una función que "da la vuelta" al valor preexistente en la memoria.

El diseño de los **ejercicios 8, 9 y 10** se basó en la perspectiva de Bildung y el modelo del Hybrid Interaction System (HIS) (Capítulo 2), donde la interacción con el autómata busca que el estudiante pase de asignar valores y expresiones simples a escribir instrucciones que logren comportamientos de mayor complejidad, que involucran sistemas lógicos y numéricos. Por ejemplo, si el estudiante modifica la batería con un valor distinto de cero, el robot sigue encendido, visualizando el fracaso e induciendo a la reflexión sobre las razones.

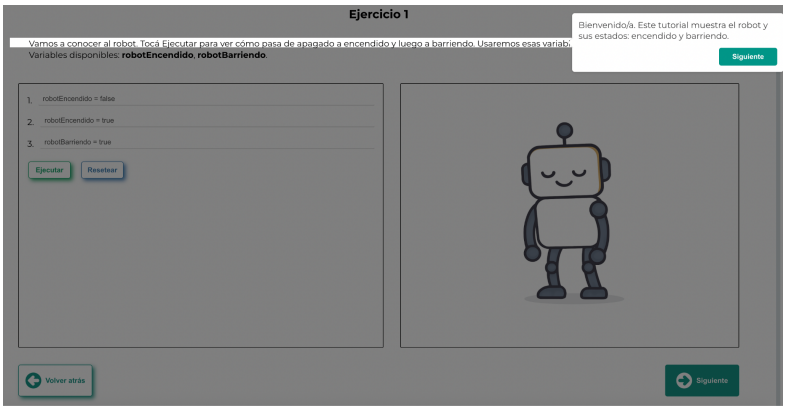
En el **ejercicio 8** se plantean desafíos que involucran expresiones compuestas que integran la conjunción (AND) y la negación (NOT) y en el **10** con múltiples apariciones del operador OR. El objetivo es coordinar múltiples condiciones para que el robot realice su tarea de limpieza.

El **ejercicio 9** introduce una capa adicional de complejidad al integrar esquemas de diferentes dominios: el numérico y el lógico. A diferencia de los retos previos que solo manipulan estados booleanos, aquí el éxito del sistema depende de una comparación relacional coordinada con múltiples condiciones.

Los **ejercicios 11 a 14** presentan situaciones que combinan el uso de los operadores vistos anteriormente. La estrategia didáctica está guiada por el principio teórico de la generalización constructiva por el cual el estudiante debe adaptar su conocimiento construido para tomar en cuenta las nuevas condiciones de los desafíos que enfrenta.

Los **ejercicios 12, 13 y 14** se encuentran en el anexo ya que no forman parte del conjunto final de ejercicios que seleccionamos para Mileva 2. Si bien nos parecieron sumamente interesantes en el capítulo 5.5 ahondaremos en el por qué de la exclusión de estos ejercicios.

3.3.1. Ejercicio 1



• **Enunciado y Desafío:** El sistema presenta al robot en estado inicial *"dormido"*. La consigna invita al usuario a presionar "Ejecutar"para observar la secuencia de estados.

• **Estado Inicial del Código:** Al ser un ejercicio introductorio el panel de instrucciones contiene tres sentencias pre-cargadas que se ejecutan secuencialmente:

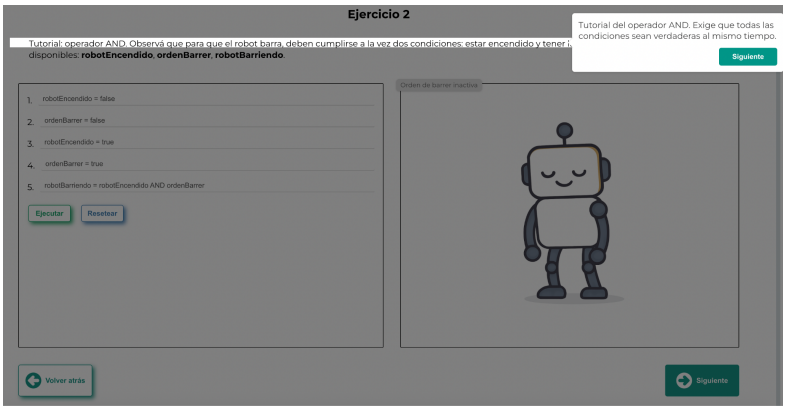
- robotEncendido = false (Estado inicial de reposo).
- robotEncendido = true (El robot abre los ojos y se activa).
- robotBarriendo = true (El robot comienza su tarea física).

Al ejecutar, Mileva resalta la línea activa en verde, permitiendo que el estudiante asocie el tiempo de procesamiento con la transformación visual del robot.

Se busca lograr que el estudiante identifique la relación causal entre los valores de las variables (true/false) y los cambios físicos en el robot (apagado, encendido, barriendo), así como también introducir la sintaxis de asignación y la naturaleza secuencial del autómata en este nuevo contexto visual.

El sujeto se concentra en objetos aislados (la variable robotEncendido o robotBarriendo) y en cómo una asignación directa modifica sus estados. Todavía no se exige la coordinación de múltiples operadores (AND/OR), sino el reconocimiento del estado booleano como una cualidad intrínseca del robot.

3.3.2. Ejercicio 2



• **Enunciado:** El sistema presenta al robot y explica que para que pueda cumplir su tarea deben cumplirse dos condiciones: estar encendido y tener la orden de barrer.

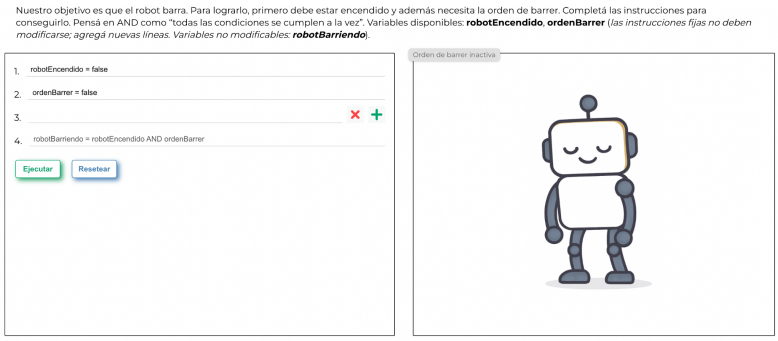
• **Estado del Código:** El panel muestra una secuencia de cinco instrucciones no editables ya que este es un ejercicio introductorio:

1. robotEncendido = false
2. ordenBarrer = false
3. robotEncendido = true
4. ordenBarrer = true
5. robotBarriendo = robotEncendido AND ordenBarrer

Al presionar “Ejecutar”, Mileva resalta secuencialmente cada línea en verde. El estudiante observa cómo se preparan los datos de entrada (líneas 1-4) y cómo, finalmente, el autómata procesa la relación lógica en la línea 5 para activar la animación del robot barriendo.

Se busca lograr que el estudiante comprenda la semántica del operador AND como una relación de necesidad conjunta. El sujeto debe identificar que para que una variable de estado final (el robot barriendo) sea true, no basta con una acción aislada, sino que deben coexistir dos estados de memoria “positivos” simultáneamente.

3.3.3. Ejercicio 3



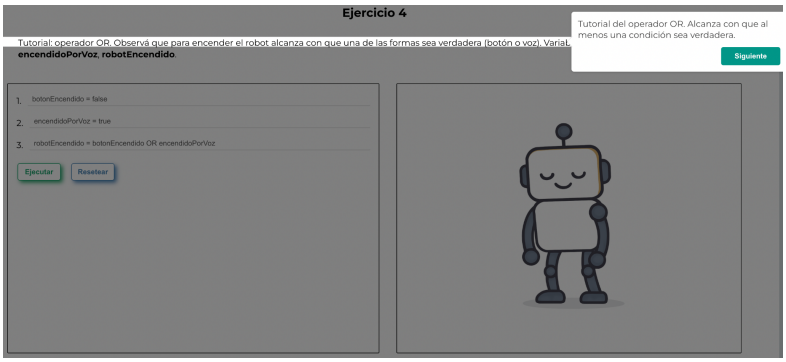
• **Enunciado y Desafío:** El sistema presenta al robot inicialmente inactivo. La consigna indica: *"Nuestro objetivo es que el robot barra. Para lograrlo, primero debe estar encendido y además necesita la orden de barrer. Completá las instrucciones para conseguirlo"*.

• **Estado Inicial del Código:** El panel muestra instrucciones fijas que establecen un estado de fallo inicial:

1. robotEncendido = false
2. ordenBarrer = false
3. (Espacio vacío para la intervención del estudiante)
4. robotBarriendo = robotEncendido AND ordenBarrer (Instrucción de evaluación fija).

• **Resolución:** El estudiante debe utilizar el botón + para agregar las sentencias necesarias que cambien el valor de las variables de entrada (por ejemplo, robotEncendido = true y ordenBarrer = true). Al presionar "Ejecutar", Mileva procesa la secuencia y validará si el resultado de la conjunción activa la animación del robot barriendo.

3.3.4. Ejercicio 4



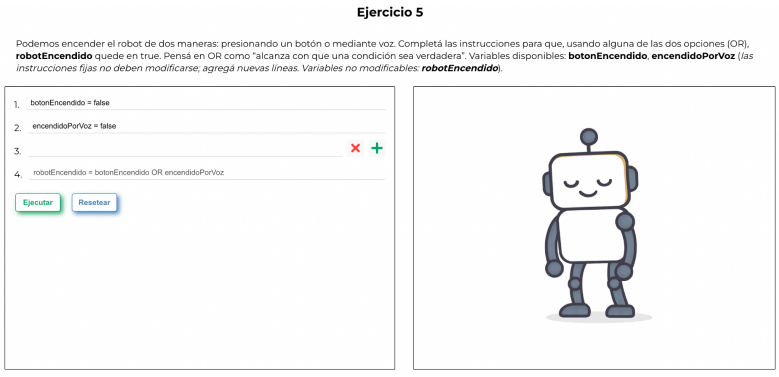
• **Enunciado:** El sistema presenta al robot y explica que puede encenderse de dos maneras: presionando un botón o mediante comandos de voz.

• **Estado del Código:** El panel muestra tres instrucciones fijas y no editables ya que se trata de un ejercicio introductorio:

1. botonEncendido = false
2. encendidoPorVoz = true
3. robotEncendido = botonEncendido OR encendidoPorVoz

Al presionar "Ejecutar", Mileva resalta secuencialmente cada línea en verde. El estudiante observa que, aunque la primera condición es false, el autómata valida la expresión completa como true debido a la segunda variable, gatillando la animación que despierta al robot.

3.3.5. Ejercicio 5



• **Enunciado y Desafío:** El sistema explica que el robot puede encenderse de dos maneras independientes. El estudiante debe completar las instrucciones para que, usando alguna de las opciones, la variable robotEncendido quede en true.

A diferencia del ejercicio anterior, el sujeto ya no es un espectador del proceso, sino que debe configurar el estado de la memoria para satisfacer una regla lógica de necesidad conjunta preestablecida. El estudiante deja de intentar adivinar el resultado y comienza a **predecir** el comportamiento del sistema basándose en la comprensión del operador **AND (ej 2)** y **el OR (ej 4)** como estructuras lógicas que pueden combinarse de distintas maneras para obtener el éxito.

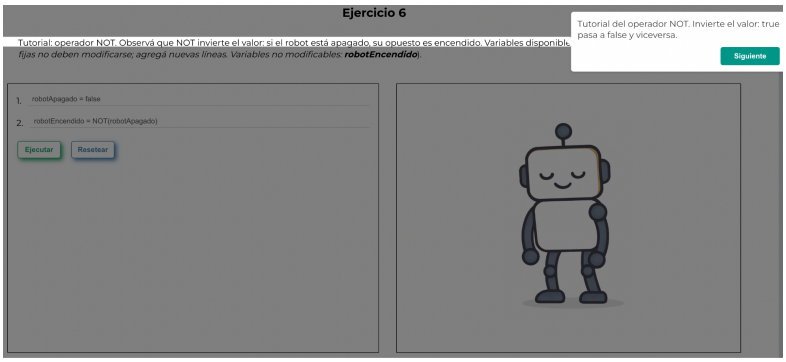
• **Estado Inicial del Código:** El panel muestra variables de entrada inicializadas en un estado de "apagado":

1. botonEncendido = false
2. encendidoPorVoz = false
3. (Espacio editable para la intervención del estudiante)
4. robotEncendido = botonEncendido OR encendidoPorVoz (Instrucción de evaluación fija).

• **Resolución:** El estudiante debe agregar las sentencias necesarias (ej. botonEncendido = true) para modificar el estado de la memoria. El sistema permite probar combinaciones (solo botón, solo voz o ambos) para verificar que el robot se activa en cualquiera de esos casos.

En este ejercicio la **intervención activa** del estudiante en el entorno del robot barrendero para el operador disyuntivo OR se plantea mediante valores de las variables de entrada editables, lo que permite al estudiante experimentar con las condiciones de éxito de la expresión lógica. Se busca que el estudiante comprenda que el operador OR persiste en un estado de veracidad siempre que al menos una de sus entradas sea verdadera, permitiendo que experimente con la suficiencia lógica y con los roles específicos de las variables botonEncendido y encendidoPorVoz que alimentan a un autómata.

3.3.6. Ejercicio 6



• **Enunciado:** El sistema presenta al robot barrendero y explica que el operador NOT invierte el valor de una variable: si el robot está apagado, su opuesto es encendido.

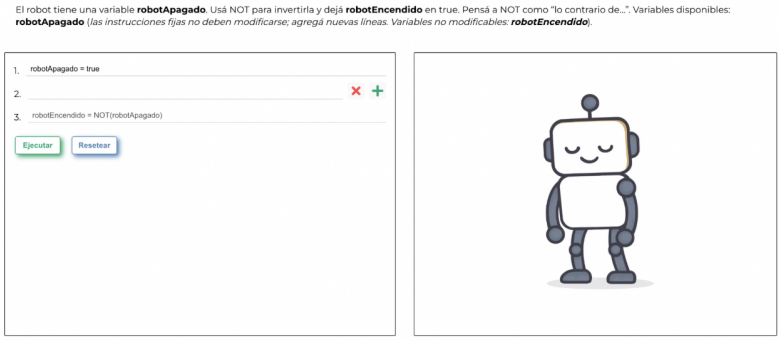
• **Estado del Código:** El panel muestra dos instrucciones fijas y no editables ya que es un ejercicio introductorio:

1. robotApagado = false
2. robotEncendido = NOT(robotApagado)

Al presionar "Ejecutar", Mileva resalta secuencialmente cada línea en verde. El estudiante observa que, aunque no se asigna true directamente a la variable de encendido, el autómata procesa la inversión de la variable robotApagado cuando llega a la sentencia final, desplegando la animación que activa al robot.

Este ejercicio introduce al estudiante en la semántica del operador NOT (negación) mediante un formato tutorial de observación guiada.

3.3.7. Ejercicio 7



• **Enunciado y Desafío:** El robot tiene una variable robotApagado inicializada en true. El estudiante debe usar el operador NOT para invertirla y lograr que robotEncendido quede en true. La consigna refuerza la noción de que NOT es "lo contrario de...".

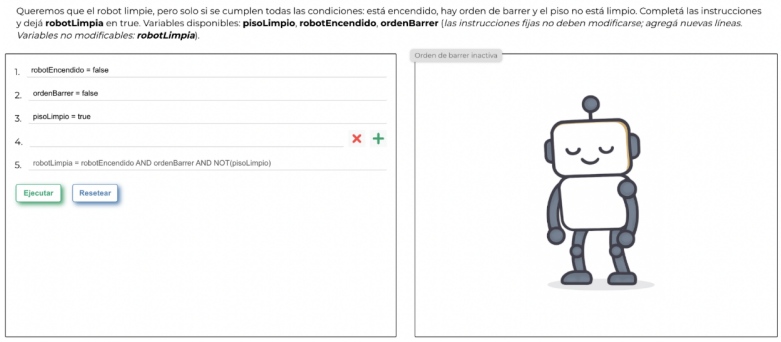
• **Estado Inicial del Código:** El panel presenta una configuración que requiere intervención en el flujo secuencial:

1. robotApagado = true
2. (Espacio vacío para la instrucción del estudiante)
3. robotEncendido = NOT(robotApagado) (Instrucción de evaluación fija).

• **Resolución:** El estudiante debe usar el botón + para agregar la instrucción (o utilizar la instruccion en blanco) que permita al autómata resolver la expresión de la línea 3.

En suma, **en los ejercicios 6 y 7**, al igual que los tutoriales de los operadores AND y OR, se presentan instrucciones fijas para que el sujeto actúe como un observador del proceso de transformación lógica que realiza el autómata, buscando que comprenda la semántica del operador NOT como una herramienta de inversión de estado, donde la acción de la computadora es procesar una función que "da la vuelta" al valor preexistente en la memoria.

3.3.8. Ejercicio 8



• **Enunciado y Desafío:** El sistema explica que el robot limpia solo si se cumplen tres condiciones: estar encendido, tener la orden de barrer y que el piso **no** esté limpio. La consigna indica: *"Completá las instrucciones y dejá robotLimpia en true"*.

• **Expresión Lógica a Evaluar:** robotLimpia = robotEncendido AND ordenBarrer AND NOT(pisoLimpio).

• **Estado Inicial del Código:** El panel presenta la siguiente configuración inicial:

1. robotEncendido = false

2. ordenBarrer = false

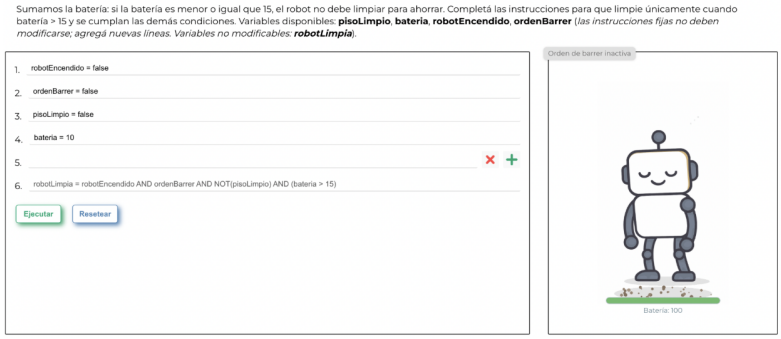
3. pisoLimpio = true (Este es el obstáculo clave: el robot no limpiará si el piso ya está limpio).

• **Resolución:** El estudiante debe realizar una intervención triple: agregar instrucciones para encender al robot (true), darle la orden (true) y, fundamentalmente, cambiar el estado del piso a false. Al ejecutar, Mileva resalta la secuencia en verde, validando si la combinación final satisface la expresión de la línea 5.

A diferencia de los retos anteriores, el estudiante ya no trabaja con un operador aislado, sino que debe gestionar una expresión compuesta que integra la conjunción (AND) y la negación (NOT). El objetivo es coordinar múltiples condiciones para que el robot realice su tarea de limpieza, lo que enfrenta al estudiante a una situación conectada con el concepto de precedencia lógica y coordinación múltiple de varios estados, incluyendo la inversión de uno de ellos.

Se busca que el estudiante experimente que la programación exige una coordinación precisa donde el éxito computacional no es un evento azaroso, sino el resultado de configurar un entorno de datos que sea coherente con reglas formales complejas.

3.3.9. Ejercicio 9



• **Enunciado y Desafío:** El sistema explica que para ahorrar energía, el robot no debe limpiar si la batería es menor o igual a 15. El estudiante debe completar las instrucciones para que la variable robotLimpia sea true únicamente cuando bateria $>$ 15 y se cumplan las condiciones de encendido, orden y suciedad del piso.

• **Expresión Lógica a Evaluar:** robotLimpia = robotEncendido AND ordenBarrer AND NOT(pisoLimpio) AND (bateria $>$ 15).

• **Estado Inicial del Código:** El panel presenta una configuración de falla múltiple:

1. robotEncendido = false

2. ordenBarrer = false

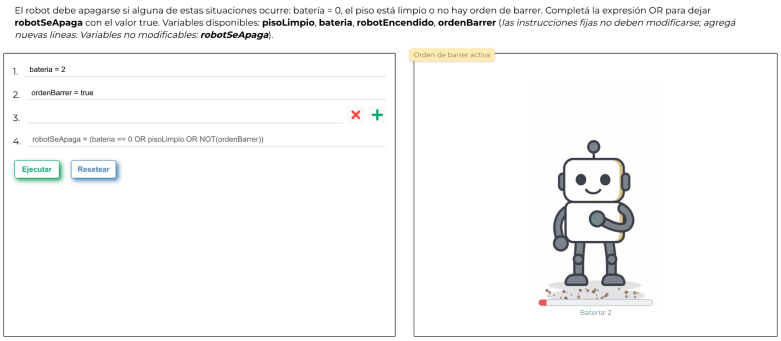
3. pisoLimpio = false

4. bateria = 10 (Este valor es el obstáculo numérico principal).

• **Resolución:** El estudiante debe intervenir activamente agregando líneas de código para encender al robot, darle la orden y, de manera crítica, aumentar el valor de la batería por encima de 15.

En suma, en el **ejercicio 9** se busca que el estudiante experimente una situación donde una variable cuantitativa (bateria) condiciona la acción global del sistema a través de una expresión lógica compuesta, así como la experimentación con comparaciones numéricas dentro de cadenas de operadores booleanos para modelar restricciones de eficiencia (ahorro de energía).

3.3.10. Ejercicio 10



• **Enunciado y Desafío:** El sistema explica que el robot debe apagarse si ocurre alguna de estas tres situaciones: la batería es igual a 0, el piso ya está limpio o no hay una orden de barrer activa. La consigna indica: *"Completá la expresión OR para dejar robotSeApaga con el valor true"*.

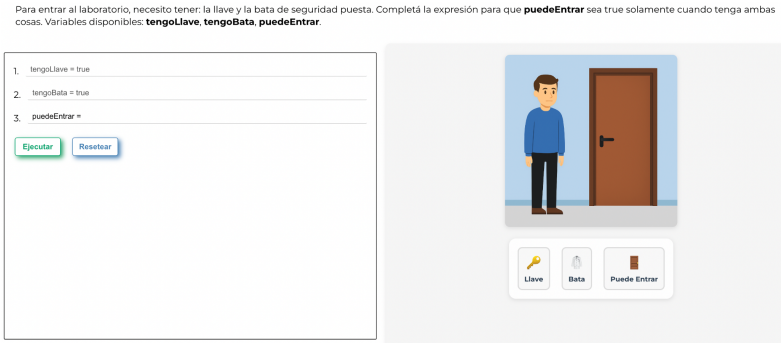
• **Expresión Lógica a Evaluar:** robotSeApaga = (bateria == 0 OR pisoLimpio OR NOT(ordenBarrer)).

• **Resolución:** El panel de código presenta variables inicializadas en un estado que mantiene al robot encendido (ej. bateria = 2, ordenBarrer = true). El estudiante debe intervenir activamente en los datos de entrada o completar la instrucción para que el sistema procese la expresión y active la animación de apagado. Se espera que el alumno experimente activando cada caso de forma independiente para validar que cualquiera de ellos dispara la acción de la máquina.

Este ejercicio representa un nivel avanzado en la comprensión de sistemas de control reactivo dentro de Mileva. A diferencia de los retos anteriores que se centraban en la activación de funciones, este desafío introduce la lógica de interrupción o seguridad, donde el éxito (definido como el apagado correcto del

autómata) depende de la detección de una falla, la finalización de una tarea o la falta de recursos en un entorno de múltiples condiciones.

3.3.11. Ejercicio 11



- **Enunciado y Desafío:** El sistema presenta a un personaje frente a la puerta de un laboratorio. La consigna indica: *"Para entrar al laboratorio, necesito tener: la llave y la bata de seguridad puesta. Completá la expresión para que puedeEntrar sea true solamente cuando tenga ambas cosas"*.
- **Estado Inicial del Código:** El panel presenta una configuración donde los requisitos previos ya están satisfechos en la memoria:

1. tengoLlave = true
2. tengoBata = true
3. puedeEntrar = (Espacio vacío para completar la expresión lógica).

- **Resolución:** El estudiante debe completar la línea 3 integrando las variables de entrada con el operador de conjunción. Al ejecutar, Mileva procesa la expresión; si el resultado es true, se visualiza la acción física: la puerta se abre y el personaje puede ingresar, validando el esquema mental del usuario. Se cuenta con restricciones que impiden asignar el valor directamente a puedeEntrar lo cual obliga al estudiante a colocar la expresión booleana correspondiente para que pueda completar el ejercicio exitosamente. Se acepta tanto tengoLlave AND tengoBata asi como tambien tengoBata AND tengoLlave .

A diferencia de los retos anteriores, que se centraban en la manipulación de objetos directos (bombitas, puertas) o autómatas (robots), este desafío utiliza una analogía del mundo físico (el acceso a un entorno restringido) para proyectar las acciones del estudiante desde el plano de la acción directa hacia el plano del pensamiento.

Se busca que el estudiante utilice variables booleanas para representar de forma abstracta las condiciones de éxito en un entorno simulado. El propósito es

consolidar la formulación de expresiones compuestas (puedeEntrar = tengoLlave AND tengoBata) como una ley que gobierna la realidad del personaje.

Ya no se trata de inicializar los valores de las variables para hacer valer true o false a una expresión final, sino que ahora es tarea del estudiante definir dicha expresión final que satisface las condiciones requeridas que plantea la consigna del ejercicio.

3.4. Evaluación por circuito corto (Grupo 4):

La evaluación por circuito corto permite evaluar expresiones booleanas que utilizan los operadores AND y OR sin necesidad de evaluar todas las subexpresiones involucradas: si el primer argumento de la función AND es falso, el valor total tiene que ser falso y por lo tanto no es necesario evaluar el segundo argumento. Análogamente si el primer argumento de la función OR es verdadero, el valor total tiene que ser verdadero y no es necesario evaluar el segundo. Comprender y saber utilizar este mecanismo de evaluación constituye un aspecto crítico del concepto de la evaluación de expresiones booleanas.

Como en el resto de los grupos de ejercicios se comienza con ejercicios introductorios, los **ejercicios 1 y 2**, desplazando el foco del estudiante desde la escritura de código hacia la observación del proceso interno del sistema, centrándose en la semántica del **AND** en el **ejercicio 1** y en la del **OR** en el **ejercicio 2.**

El propósito es que el sujeto identifique bajo qué condiciones el autómata decide dejar de evaluar una parte de una expresión por considerarla ya definida.

A partir del **ejercicio 3** se exige que el estudiante intervenga activamente para lograr la evaluación de las expresiones boolenas mediante el mínimo esfuerzo computacional y refuerce su comprensión de este tipo de evaluación

El desafío obliga al sujeto a razonar no solo sobre el valor de verdad final (true), sino sobre el orden de los operandos y su impacto en el flujo de ejecución del autómata.

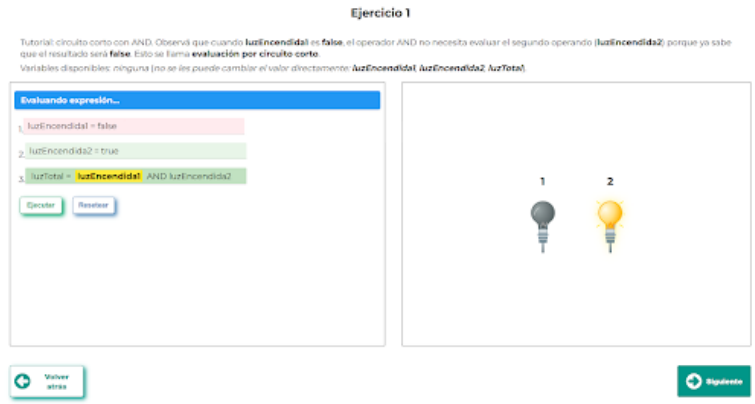
Desde la perspectiva de la interacción planteada en *Bildung* (Capítulo 2), buscamos que el estudiante observe que la computadora "se ahorra trabajo"(ignora el resto de la expresión) y experimente una transformación de su percepción preconcebida a partir de la matemática. Por ejemplo en uno de los ejercicios introductorios donde al encender la bombita 3, verá que el resaltado amarillo pasa por todos los términos antes de detenerse. Esta visualización del proceso le permite comparar las elecciones que funcionan con "lentitud"de las de la opción óptima que funcionan con "rapidez", experimentando en la interacción ambos procesos de evaluación.

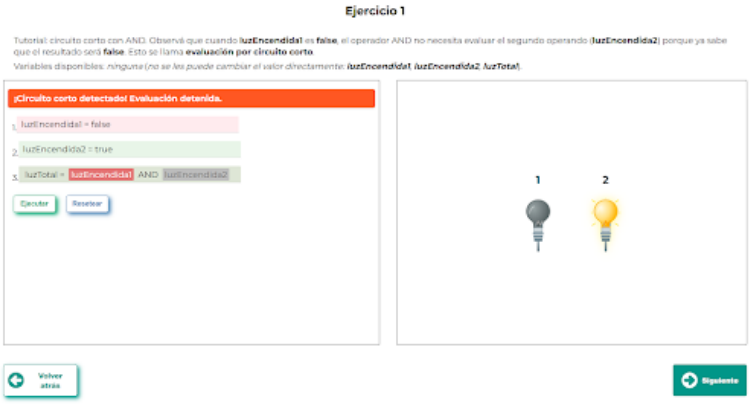
Este ejercicio refuerza la noción de que en computación el orden de los factores sí altera el proceso, aunque no altere el valor final. Este es un aspecto crítico fundamental para el desarrollo del pensamiento computacional, que se refuerza mediante la repetición de situaciones similares en los ejercicios siguientes. Entre ellos, se destaca el **ejercicio 4** que constituye el reto de mayor complejidad dentro del grupo hasta este punto, ya que integra los tres operadores lógicos fundamentales (AND, OR, NOT) en una sola expresión compuesta.

Se busca que el estudiante comprenda el funcionamiento de expresiones booleanas complejas, identificando el impacto de la precedencia de operadores (paréntesis) y cómo el circuito corto puede ocurrir tanto en el nivel global de la expresión como dentro de sus subexpresiones.

A continuación se describen los ejercicios del grupo de circuito corto en detalle.

3.4.1. Ejercicio 1





Este ejercicio introduce un concepto avanzado de la computación: la evaluación de circuito corto (*short-circuit evaluation*). Este reto inicial es puramente demostrativo, desplazando el foco del estudiante desde la escritura de código hacia la observación del proceso interno del sistema.

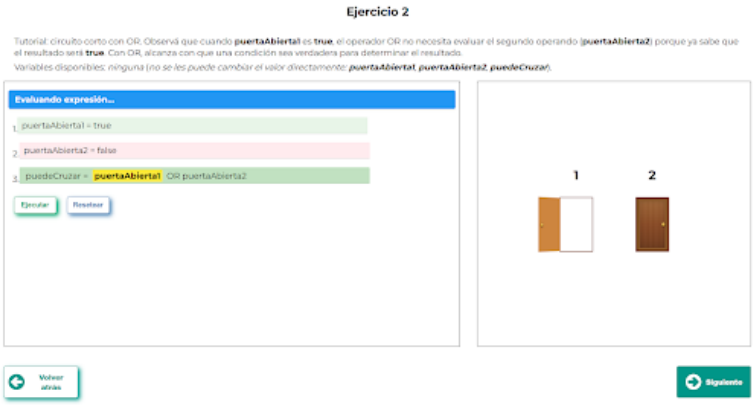
• **Enunciado y Desafío:** El sistema presenta una expresión lógica compuesta utilizando el operador AND (por ejemplo: resultado = variable1 AND

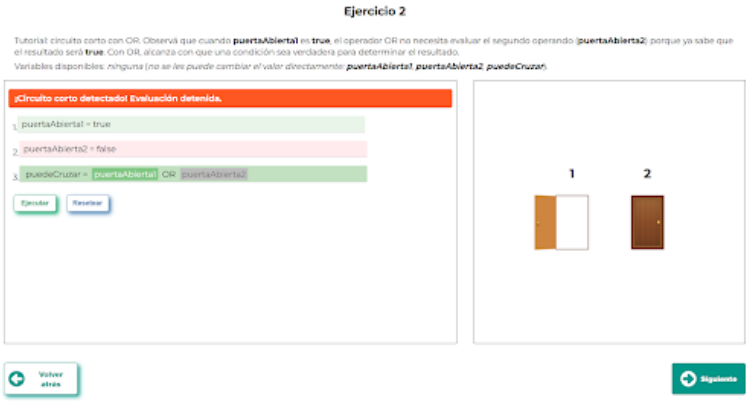
variable2). El estudiante solo debe presionar el botón “Ejecutar”para observar el comportamiento del motor de ejecución de Mileva.

- **Visualización:**
  - **Parte 1 (Evaluación Inicial):** Al iniciar, el sistema resalta en color amarillo la primera subexpresión (el operando de la izquierda). Esto indica el foco de atención actual del autómata.
  - **Parte 2 (Detección de Circuito Corto):** Si el primer operando es false, el sistema detecta que, según las reglas del AND, el resultado final será necesariamente false independientemente del segundo valor. En ese instante, Mileva muestra un cartel informativo notificando la detección del “Circuito Corto”, detiene la evaluación del resto de la línea (que nunca llega a resaltarse) y carga el valor final en la variable resultado.

Desde la perspectiva de **Bildung**, el cartel informativo y la detención de la ejecución actúan como un “momento de revelación”. El estudiante ya no solo sabe *qué* hace el programa, sino *cómo* lo hace, mediante el efecto de que la segunda parte de la expresión nunca se resalta en amarillo. Esta ausencia de acción es tan potente como la acción misma, ya que evidencia físicamente el concepto de “omisión por optimización”.

3.4.2. Ejercicio 2





Este ejercicio profundiza en el concepto de evaluación optimizada, desplazando ahora el foco hacia el operador lógico **OR**. Al igual que el ejercicio anterior del grupo, tiene un carácter puramente **demostrativo**, diseñado para que el estudiante sea testigo del proceso de toma de decisiones interno del autómata cuando una condición de suficiencia lógica se cumple.

• **Enunciado y Desafío:** El sistema presenta una situación con dos puertas y una variable de estado global puedeCruzar. La lógica depende de si al menos una de las dos puertas está abierta. El estudiante simplemente debe presionar "Ejecutar"para observar el motor de ejecución en acción.

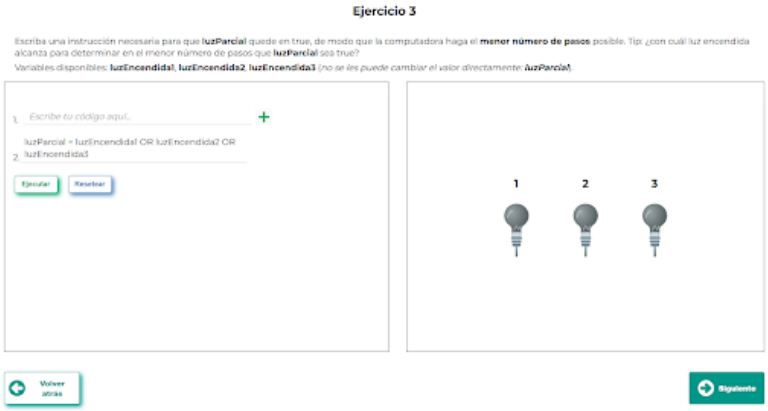
• **Visualización:**

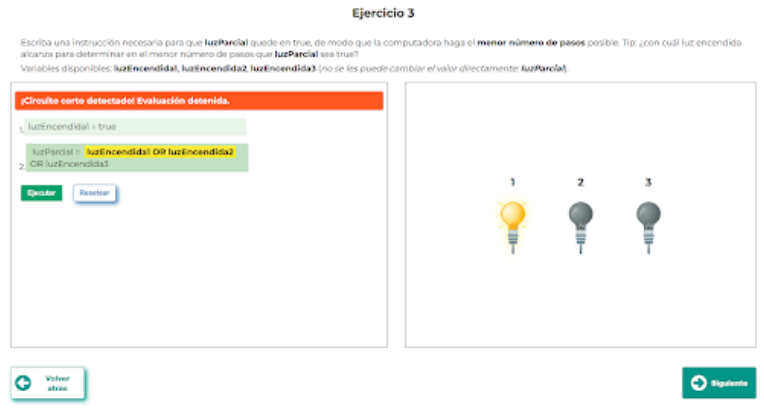
◦ **Parte 1 (Evaluación Inicial):** En el panel de código, la instrucción de la línea 3 resalta en **amarillo** la primera subexpresión: puertaAbierta1. Esto indica que el autómata está procesando ese operando en particular.

◦ **Parte 2 (Detección de Circuito Corto):** Dado que puertaAbierta1 es true, el sistema detecta instantáneamente que no necesita conocer el estado de la segunda puerta para validar que el usuario puedeCruzar. En ese momento, aparece un cartel naranja indicando "¡Circuito corto detectado! Evaluación detenida", y el resto de la expresión (OR puertaAbierta2) queda visualmente opacado, indicando que fue ignorado por el computador.

Desde la perspectiva del Bildung el banner naranja que indica que se detectó circuito corto y que la evaluación se detuvo actúa, al igual que la información del ejercicio anterior, como un "momento de toma de conciencia"provocado por el sistema al resaltar el error o la detención de forma explícita.

3.4.3. Ejercicio 3





Ejercicio 3

Escriba una instrucción necesaria para que **luzParcial** quede en true, de modo que la computadora haga el **menor número de pasos** posible. Tip: ¿con cuál luz encendida alcanza para determinar en el menor número de pasos que **luzParcial** sea true?

Variables disponibles: **luzEncendida1**, **luzEncendida2**, **luzEncendida3** *(no se les puede cambiar el valor directamente: **luzParcial**)*.

¡Circuito corto detectado! Evaluación detenida.

```
1 luzEncendida1 = true
2 luzParcial = luzEncendida1 OR luzEncendida2 OR luzEncendida3
```

Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 3

Escriba una instrucción necesaria para que **luzParcial** quede en true, de modo que la computadora haga el **menor número de pasos** posible. Tip: ¿con cuál luz encendida alcanza para determinar en el menor número de pasos que **luzParcial** sea true?

Variables disponibles: **luzEncendida1**, **luzEncendida2**, **luzEncendida3** *(no se les puede cambiar el valor directamente: **luzParcial**)*.

¡Circuito corto detectado! Evaluación detenida.

```
1 luzEncendida1 = true
2 luzParcial = luzEncendida1 OR luzEncendida2 OR luzEncendida3
```

Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 3

Escriba una instrucción necesaria para que **luzParcial** quede en true, de modo que la computadora haga el **menor número de pasos** posible. Tip: ¿con cuál luz encendida alcanza para determinar en el menor número de pasos que **luzParcial** sea true?

Variables disponibles: **luzEncendida1**, **luzEncendida2**, **luzEncendida3** *(no se les puede cambiar el valor directamente: **luzParcial**)*.

¡Circuito corto detectado! Evaluación detenida.

```
1 luzEncendida1 = true
2 luzParcial = luzEncendida1 OR luzEncendida2 OR luzEncendida3
```

Ejecutar | Resetear

Volver atrás | Siguiente

A diferencia de los anteriores, que funcionaban como un tutorial meramente demostrativo, este reto exige que el estudiante intervenga activamente para, no solo lograr un resultado, sino de lograrlo mediante el mínimo esfuerzo computacional.

• **Enunciado y Desafío:** El sistema presenta tres bombitas apagadas y una expresión compuesta: luzParcial = luzEncendida1 OR luzEncendida2 OR luzEncendida3. La consigna es: *"Escriba la instrucción necesaria para que luzParcial sea true, de modo que la computadora haga el menor número de pasos posible"*.

• **Resolución:**

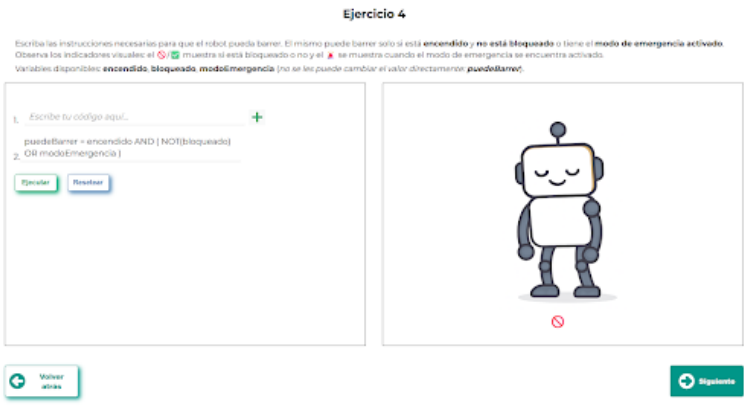
◦ **Análisis Inicial:** El estudiante observa que tiene tres variables de entrada. El "tip"lo invita a reflexionar: ¿con qué bombita encendida se alcanza el objetivo más rápido?.

◦ **Ejecución y Resaltado:** Si el estudiante inicializa luzEncendida1 = true, al presionar "Ejecutar", el sistema resalta en amarillo el primer término de la expresión (luzEncendida1) en la línea 2.

◦ **Detección de Circuito Corto:** Al ser el primer operando true, Mileva despliega un cartel naranja indicando "¡Circuito corto detectado! Evaluación detenida". Inmediatamente, los términos restantes (OR luzEncendida2 OR luzEncendida3) se muestran opacados o ignorados, cargando el valor final en la variable.

Se busca que el estudiante experimente la naturaleza secuencial y perezosa de la evaluación de la expresión booleana por circuito corto, induciendolo a razonar no solo sobre el valor de verdad final (true), sino sobre el orden de los operandos y su impacto en el flujo de ejecución del autómata.

3.4.4. Ejercicio 4



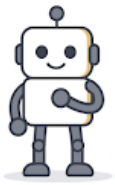
Ejercicio 4

Escriba las instrucciones necesarias para que el robot pueda barrer. El mismo puede barrer solo si está **encendido** y **no está bloqueado** o tiene el **modo de emergencia activado**.
Observa los indicadores visuales: el 🚫/✅ muestra si está bloqueado o no y el 🚨 se muestra cuando el modo de emergencia se encuentra activado.
Variables disponibles: **encendido**, **bloqueado**, **modoEmergencia** *(no se les puede cambiar el valor directamente: **puedeBarrer**)*.

Evaluando expresión...

```
1 encendido = true
2 bloqueado = false
3 puedeBarrer = encendido AND ( NOT(bloqueado) OR modoEmergencia )
```

Ejecutar | Resetear



Volver atrás | Siguiente

Ejercicio 4

Escriba las instrucciones necesarias para que el robot pueda barrer. El mismo puede barrer solo si está **encendido** y **no está bloqueado** o tiene el **modo de emergencia activado**.
Observa los indicadores visuales: el 🚫/✅ muestra si está bloqueado o no y el 🚨 se muestra cuando el modo de emergencia se encuentra activado.
Variables disponibles: **encendido**, **bloqueado**, **modoEmergencia** *(no se les puede cambiar el valor directamente: **puedeBarrer**)*.

Evaluando expresión...

```
1 encendido = true
2 bloqueado = false
3 puedeBarrer = encendido AND ( NOT( bloqueado ) OR modoEmergencia )
```

Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 4

Escriba las instrucciones necesarias para que el robot pueda barrer. El mismo puede barrer solo si está **encendido** y **no está bloqueado** o tiene el **modo de emergencia activado**.
Observa los indicadores visuales: el 🚫/✅ muestra si está bloqueado o no y el 🚨 se muestra cuando el modo de emergencia se encuentra activado.
Variables disponibles: **encendido**, **bloqueado**, **modoEmergencia** *(no se les puede cambiar el valor directamente: **puedeBarrer**)*.

Evaluando expresión...

```
1 encendido = true
2 bloqueado = false
3 puedeBarrer = encendido AND ( NOT( bloqueado ) OR modoEmergencia )
```

Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 4

Escriba las instrucciones necesarias para que el robot pueda barrer. El mismo puede barrer solo si está **encendido** y **no está bloqueado** o tiene el **modo de emergencia activado**.
Observa los indicadores visuales: el 🚫/✅ muestra si está bloqueado o no y el 🚨 se muestra cuando el modo de emergencia se encuentra activado.
Variables disponibles: **encendido**, **bloqueado**, **modoEmergencia** (*no se les puede cambiar el valor directamente*: ***puedeBarrer***).

Evaluando expresión...

```
1 encendido = true
2 bloqueado = false
3 puedeBarrer = encendido AND ( NOT(bloqueado) OR modoEmergencia )
```

Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 4

Escriba las instrucciones necesarias para que el robot pueda barrer. El mismo puede barrer solo si está **encendido** y **no está bloqueado** o tiene el **modo de emergencia activado**.
Observa los indicadores visuales: el 🚫/✅ muestra si está bloqueado o no y el 🚨 se muestra cuando el modo de emergencia se encuentra activado.
Variables disponibles: **encendido**, **bloqueado**, **modoEmergencia** (*no se les puede cambiar el valor directamente*: ***puedeBarrer***).

¡Circuito corto detectado! Evaluación detenida.

```
1 encendido = true
2 bloqueado = false
3 puedeBarrer = encendido AND ( NOT(bloqueado) OR modoEmergencia )
```

Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 4

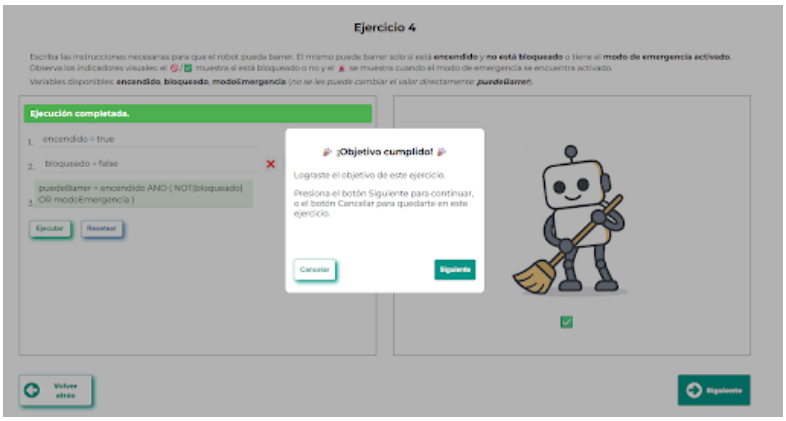
Escriba las instrucciones necesarias para que el robot pueda barrer. El mismo puede barrer solo si está **encendido** y **no está bloqueado** o tiene el **modo de emergencia activado**.
Observa los indicadores visuales: el 🚫/✅ muestra si está bloqueado o no y el 🚨 se muestra cuando el modo de emergencia se encuentra activado.
Variables disponibles: **encendido**, **bloqueado**, **modoEmergencia** (*no se les puede cambiar el valor directamente*: ***puedeBarrer***).

¡Circuito corto detectado! Evaluación detenida.

```
1 encendido = true
2 bloqueado = false
3 puedeBarrer = encendido AND ( NOT(bloqueado) OR modoEmergencia )
```

Ejecutar | Resetear

Volver atrás | Siguiente



Este ejercicio constituye el reto de mayor complejidad dentro del grupo hasta este punto, ya que integra los tres operadores lógicos fundamentales (AND, OR, NOT) en una sola expresión compuesta. El estudiante debe coordinar múltiples estados para permitir que un robot cumpla su tarea, mientras observa cómo el autómata optimiza la evaluación mediante el circuito corto.

• **Enunciado y Desafío:** El sistema presenta un robot y tres indicadores visuales (encendido, bloqueo y emergencia). La consigna indica: *"Escriba las instrucciones necesarias para que el robot pueda barrer. El mismo puede barrer solo si está encendido y no está bloqueado o tiene el modo de emergencia activado"*.

• **Expresión Lógica a Evaluar:** puedeBarrer = encendido AND (NOT(bloqueado) OR modoEmergencia)

Análisis del Proceso de Ejecución

Se visualiza el ciclo de interacción recíproca de la siguiente manera (para explicarlo de la mejor forma posible se asume que el estudiante ingresa la solución óptima: encendido = true y bloqueado = false):

1. **Planteo:** El estudiante recibe el panel con las variables disponibles y la expresión lógica. Debe decidir qué valores asignar para que puedeBarrer sea true.

2. **Evaluación del primer término:** Al ejecutar, el sistema resalta en **amarillo** la variable encendido. Si es true, el autómata continúa, ya que el AND aún requiere evaluar el segundo término (el paréntesis).

3. **Evaluación anidada:** El foco amarillo entra al paréntesis y evalúa bloqueado. Al estar dentro de un NOT, el sistema procesa la inversión del valor.

4. **Detección de Circuito Corto:** Si NOT(bloqueado) resulta en true, el operador OR dentro del paréntesis detecta que ya no necesita evaluar modoEmergencia. En este punto, aparece el cartel naranja de "¡Circuito corto detectado! Evaluación detenida". La variable modoEmergencia aparece opacada y el estudiante visualiza que la misma no se llegó a evaluar gracias al circuito corto.

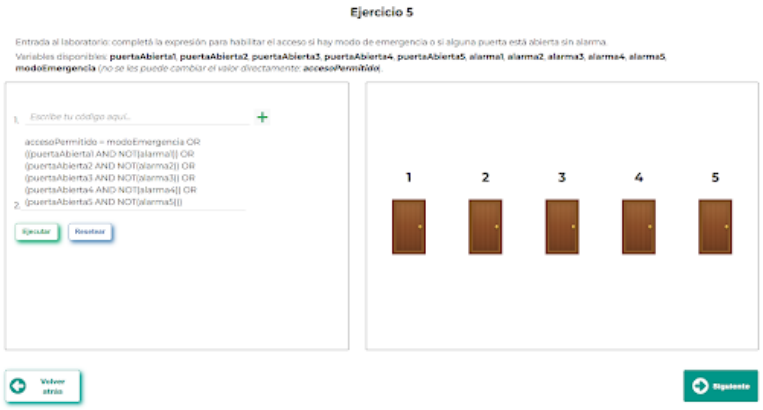
5. **Visualización del Efecto:** Una vez resuelta la lógica, se visualiza la **acción física**: el robot comienza a barrer. Esta es la "visualización del efecto"que valida el razonamiento del estudiante.

6. **Feedback de Éxito:** Aparece el mensaje "¡Objetivo cumplido!"

En caso de que el estudiante no ingrese la solución óptima, siempre y cuando las instrucciones ingresadas produzcan que puedeBarrer sea true, el sistema mostrará un cartel de éxito, aunque no se haya sacado provecho al concepto de circuito corto. Esto es porque en este caso, el estudiante notará que se evalúa toda la expresión, dándose cuenta de que su solución, si bien es correcta, no fue óptima lo cual, se espera que lo motive a intentar hallar la solución óptima.

En cualquier caso, en el último ejercicio de este grupo se intentará sintetizar la característica principal de la semántica de circuito corto de la evaluación de expresiones booleanas, ya que el estudiante no tendrá otra opción más que encontrar la solución óptima para completar exitosamente el ejercicio. Para ello, se plantea el desafío de la anticipación del flujo de ejecución y la evaluación no solo del valor de verdad, sino también del costo computacional de cada decisión lógica en una estructura jerárquica de expresiones.

3.4.5. Ejercicio 5



Este ejercicio representa, junto con el siguiente ejercicio, el cierre del bloque de circuito corto y busca una aproximación al concepto de evaluación por circuito corto, mediante la consolidación de las experiencias previas. A diferencia de los desafíos anteriores, la estructura de la expresión lógica está diseñada de tal manera que la solución más eficiente es evidente desde el inicio del código. En este caso, el hecho de que la penalización por no elegir la solución óptima pueda llegar a ser muy costosa (dependiendo de qué puerta elije abrir el estudiante) en lo que a tiempo y cómputo computacional refiere, busca inducir al estudiante a buscar una solución superadora.

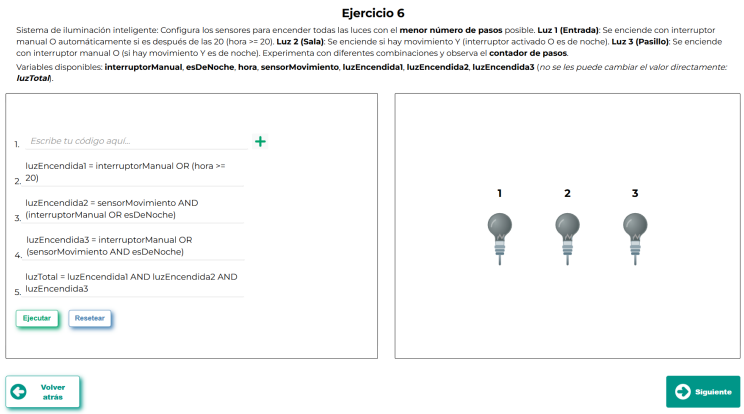
En adición, se busca que el estudiante elija de forma intencional el camino de ejecución que requiera el menor procesamiento por parte del autómata, reforzando la idea de que el orden de los términos en una cadena de OR es una decisión de diseño crítica para la eficiencia.

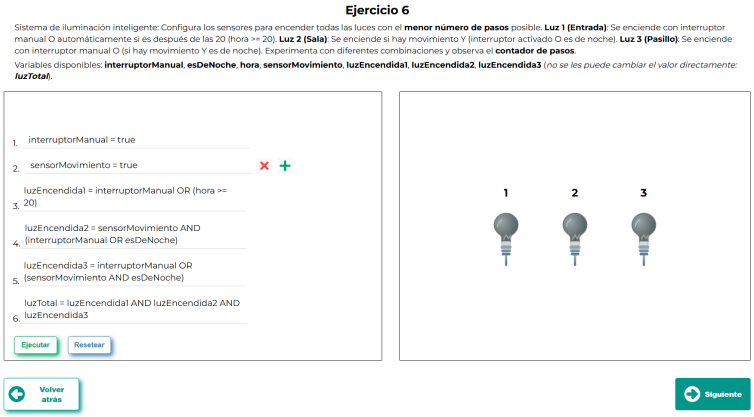
• **Enunciado y Desafío:** El sistema presenta cinco puertas y un sensor de emergencia. El objetivo es habilitar el accesoPermitido. La regla indica que se

puede entrar si hay "modo de emergencia" **O** si alguna de las cinco puertas está abierta y su respectiva alarma desactivada.

- **Expresión Lógica a Evaluar:** accesoPermitido = modoEmergencia OR ((puertaAbierta1 AND NOT(alarma1)) OR ... OR (puertaAbierta5 AND NOT(alarma5)))
- **Resolución:** El estudiante debe inicializar las variables necesarias. Al estar modoEmergencia al principio de la cadena de disyunciones (OR), asignar modoEmergencia = true se presenta como la opción óptima. Al ejecutar la solución óptima, el sistema resaltará en amarillo únicamente ese primer término y, de forma inmediata, detendrá la evaluación con el cartel naranja informativo, dejando el resto de la compleja expresión sin procesar.

3.4.6. Ejercicio 6





Ejercicio 6

Sistema de iluminación inteligente: Configura los sensores para encender todas las luces con el **menor número de pasos** posible. **Luz 1 (Entrada)**: Se enciende con interruptor manual O automáticamente si es después de las 20 (hora >= 20). **Luz 2 (Sala)**: Se enciende si hay movimiento Y (interruptor activado O es de noche). **Luz 3 (Pasillo)**: Se enciende con interruptor manual O (si hay movimiento Y es de noche). Experimenta con diferentes combinaciones y observa el **contador de pasos**.

Variables disponibles: **interruptorManual**, **esDeNoche**, **hora**, **sensorMovimiento**, **luzEncendida1**, **luzEncendida2**, **luzEncendida3** (*no se les puede cambiar el valor directamente:* ***luzTotal***).

¡Circuito corto detectado! Evaluación detenida. Evaluaciones: 1

```
1. interruptorManual = true
2. sensorMovimiento = true
3. luzEncendida1 = interruptorManual OR (hora >= 20)
4. luzEncendida2 = sensorMovimiento AND (interruptorManual OR esDeNoche)
5. luzEncendida3 = interruptorManual OR (sensorMovimiento AND esDeNoche)
6. luzTotal = luzEncendida1 AND luzEncendida2 AND luzEncendida3
```

Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 6

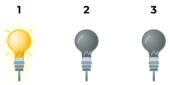
Sistema de iluminación inteligente: Configura los sensores para encender todas las luces con el **menor número de pasos** posible. **Luz 1 (Entrada)**: Se enciende con interruptor manual O automáticamente si es después de las 20 (hora >= 20). **Luz 2 (Sala)**: Se enciende si hay movimiento Y (interruptor activado O es de noche). **Luz 3 (Pasillo)**: Se enciende con interruptor manual O (si hay movimiento Y es de noche). Experimenta con diferentes combinaciones y observa el **contador de pasos**.

Variables disponibles: **interruptorManual**, **esDeNoche**, **hora**, **sensorMovimiento**, **luzEncendida1**, **luzEncendida2**, **luzEncendida3** (*no se les puede cambiar el valor directamente:* ***luzTotal***).

Evaluando expresión... Evaluaciones: 2

```
1. interruptorManual = true
2. sensorMovimiento = true
3. luzEncendida1 = interruptorManual OR (hora >= 20)
4. luzEncendida2 = sensorMovimiento AND ( interruptorManual OR esDeNoche)
5. luzEncendida3 = interruptorManual OR (sensorMovimiento AND esDeNoche)
6. luzTotal = luzEncendida1 AND luzEncendida2 AND luzEncendida3
```

Ejecutar | Resetear



Volver atrás | Siguiente

Ejercicio 6

Sistema de iluminación inteligente: Configura los sensores para encender todas las luces con el **menor número de pasos** posible. **Luz 1 (Entrada)**: Se enciende con interruptor manual O automáticamente si es después de las 20 (hora >= 20). **Luz 2 (Sala)**: Se enciende si hay movimiento Y (interruptor activado O es de noche). **Luz 3 (Pasillo)**: Se enciende con interruptor manual O (si hay movimiento Y es de noche). Experimenta con diferentes combinaciones y observa el **contador de pasos**.

Variables disponibles: **interruptorManual**, **esDeNoche**, **hora**, **sensorMovimiento**, **luzEncendida1**, **luzEncendida2**, **luzEncendida3** (*no se les puede cambiar el valor directamente:* ***luzTotal***).

¡Circuito corto detectado! Evaluación detenida. Evaluaciones: 3

```
1. interruptorManual = true
2. sensorMovimiento = true
3. luzEncendida1 = interruptorManual OR (hora >= 20)
4. luzEncendida2 = sensorMovimiento AND ( interruptorManual OR esDeNoche )
5. luzEncendida3 = interruptorManual OR (sensorMovimiento AND esDeNoche)
6. luzTotal = luzEncendida1 AND luzEncendida2 AND luzEncendida3
```

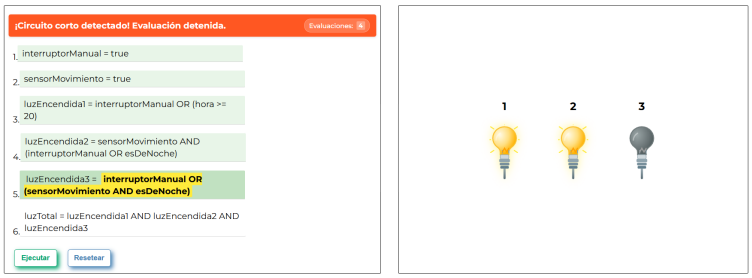
Ejecutar | Resetear

Volver atrás | Siguiente

Ejercicio 6

Sistema de iluminación inteligente: Configura los sensores para encender todas las luces con el **menor número de pasos** posible. **Luz 1 (Entrada)**: Se enciende con interruptor manual O automáticamente si es después de las 20 (hora >= 20). **Luz 2 (Sala)**: Se enciende si hay movimiento Y (interruptor activado O es de noche). **Luz 3 (Pasillo)**: Se enciende con interruptor manual O (si hay movimiento Y es de noche). Experimenta con diferentes combinaciones y observa el **contador de pasos**.

Variables disponibles: **interruptorManual**, **esDeNoche**, **hora**, **sensorMovimiento**, **luzEncendida1**, **luzEncendida2**, **luzEncendida3** (*no se les puede cambiar el valor directamente:* ***luzTotal***).



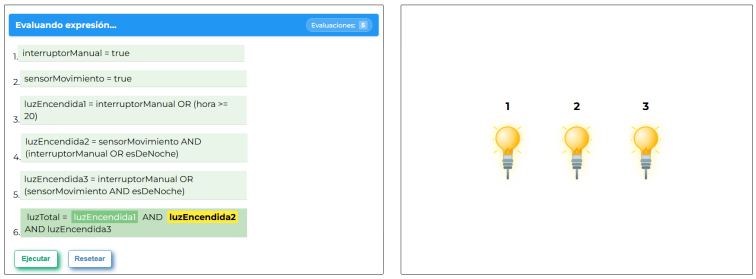
Volver atrás

Siguiente

Ejercicio 6

Sistema de iluminación inteligente: Configura los sensores para encender todas las luces con el **menor número de pasos** posible. **Luz 1 (Entrada)**: Se enciende con interruptor manual O automáticamente si es después de las 20 (hora >= 20). **Luz 2 (Sala)**: Se enciende si hay movimiento Y (interruptor activado O es de noche). **Luz 3 (Pasillo)**: Se enciende con interruptor manual O (si hay movimiento Y es de noche). Experimenta con diferentes combinaciones y observa el **contador de pasos**.

Variables disponibles: **interruptorManual**, **esDeNoche**, **hora**, **sensorMovimiento**, **luzEncendida1**, **luzEncendida2**, **luzEncendida3** (*no se les puede cambiar el valor directamente:* ***luzTotal***).



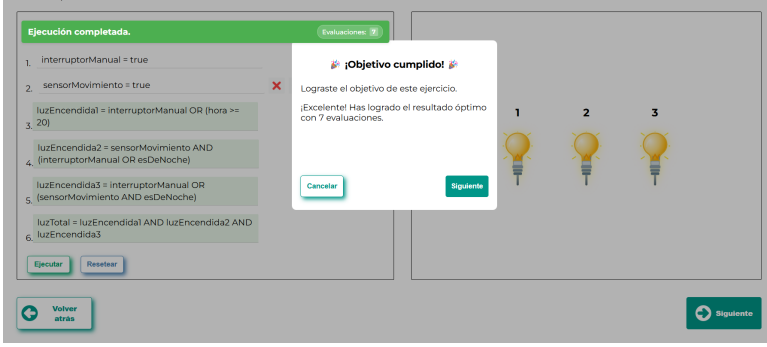
Volver atrás

Siguiente

Ejercicio 6

Sistema de iluminación inteligente: Configura los sensores para encender todas las luces con el **menor número de pasos** posible. **Luz 1 (Entrada)**: Se enciende con interruptor manual O automáticamente si es después de las 20 (hora >= 20). **Luz 2 (Sala)**: Se enciende si hay movimiento Y (interruptor activado O es de noche). **Luz 3 (Pasillo)**: Se enciende con interruptor manual O (si hay movimiento Y es de noche). Experimenta con diferentes combinaciones y observa el **contador de pasos**.

Variables disponibles: **interruptorManual**, **esDeNoche**, **hora**, **sensorMovimiento**, **luzEncendida1**, **luzEncendida2**, **luzEncendida3** (*no se les puede cambiar el valor directamente:* ***luzTotal***).



Volver atrás

Siguiente

Este ejercicio constituye definitivamente el final del bloque de ejercicios de circuito corto y representa un salto cualitativo en la complejidad del pensamiento computacional requerido. A diferencia de los anteriores, el estudiante se enfrenta a un desafío donde múltiples variables y de diferentes tipos (interruptorManual, esDeNoche, hora, sensorMovimiento) están interconectadas a través de diversas expresiones lógicas para controlar tres luces distintas. El éxito no solo depende de encender las luces, sino de hacerlo con el mínimo número de pasos, verificado por un nuevo contador de evaluaciones.

• **Enunciado y Desafío:** Configurar los sensores para encender todas las luces con el menor número de evaluaciones. Cada luz tiene su propia regla lógica (unas basadas en OR y otras en AND con paréntesis), y el estudiante debe experimentar con combinaciones hasta alcanzar el resultado óptimo (7 evaluaciones).

Análisis del Proceso de Ejecución

A través de la ejecución de este ejercicio, se observa cómo el estudiante atraviesa el ciclo de interacción recíproca:

1. **Planteo e Inicialización:** El estudiante observa las tres luces apagadas y las tres expresiones lógicas. A modo de explicar el flujo completo del ejercicio supongamos que el estudiante inicializa las variables de entrada con la solución óptima(ej. interruptorManual = true y sensorMovimiento = true).

2. **Evaluación de Luz 1:** El sistema resalta en amarillo interruptorManual. Como es true y el operador es OR, ocurre un circuito corto. El contador de evaluaciones marca 1. El estudiante visualiza que la Luz 1 se enciende inmediatamente.

3. **Evaluación de Luz 2:** El proceso es más complejo. Primero evalúa sensorMovimiento (true). Como es un AND, debe entrar al paréntesis. Dentro del paréntesis, evalúa interruptorManual. Al ser true, el OR interno se detiene por circuito corto. El contador sigue subiendo (evaluaciones 2 y 3).

4. **Evaluación de Luz 3:** Se repite la lógica de optimización. interruptorManual es true, por lo que el OR principal de la Luz 3 se detiene tras una sola evaluación (evaluación 4).

5. **Evaluación de Luz Total:** El autómata, mediantes las últimas 3 evaluaciones, verifica que las tres variables de luz sean true para confirmar el éxito del sistema global.

6. **Feedback de Éxito y Optimización:** Aparece el mensaje "¡Objetivo cumplido!". Mileva felicita al estudiante por haber logrado el resultado óptimo con 7 evaluaciones, validando su estrategia de eficiencia.

3.5. Formalización (Grupo 5)

A diferencia de los grupos de ejercicios anteriores, se despoja al estudiante de contextos visuales concretos (como luces, puertas o robots) y se introducen nombres (*a, b, c*) para representar las variables. La fundamentación teórica se basa en que una vez que el estudiante ha interactuado con variados objetos y situaciones concretas, puede ser inducido a reflexionar sobre las mismas pero en

el plano del pensamiento, despegándose de los casos concretos y aproximándose al conocimiento conceptual y formal (ver Capítulo 2).

En el **ejercicio 1** se plantean situaciones con asignaciones directas de valores y en **los ejercicios 2 y 3** el estudiante debe construir una relación de dependencia. El valor de una variable ya no es una decisión arbitraria del usuario, sino que se define como una función de los estados de otras variables, usando el operador AND en el 2 y el OR en el 3. En el **ejercicio 4** se sigue la misma pauta con el operador NOT.

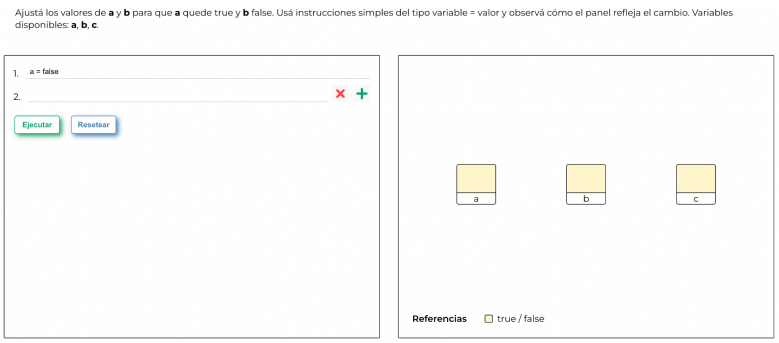
Los **ejercicios 5 y 6** introducen variables editables buscando reforzar la comprensión de que mediante la interacción el estudiante puede actuar sobre el contenido de los contenedores de entrada para determinar el estado de un tercer contenedor de salida, así como también sobre la doble semántica de la asignación, la comprensión del operador OR, la conjunción múltiple usando (NOT) y (AND) y las diferencias entre los operadores.

En el ejercicio 7 , a diferencia de los ejercicios anteriores que se centraban en operadores aislados o secuenciales, este desafío exige la gestión de una expresión lógica compuesta utilizando símbolos abstractos, integrando la negación (NOT), la conjunción (AND) y la disyunción (OR).

Dado que los ejercicios son similares tanto a los planteados anteriormente sobre variables abstractas, como a los planteados en la versión original de Mileva sobre formalización, se incluye aquí una descripción breve. Los **ejercicios 8 y 9** de este grupo se incluyen en el anexo ya que no formaron parte de la versión final de Mileva, en el capítulo 5.5 explicaremos el trasfondo detrás de esta decisión.

Cabe destacar igual que en los demás grupos, en el grupo de formalización cada ejercicio cuenta con sus propios mensajes de error en caso de que el estudiante falle en la resolución. El objetivo de dichos mensajes es ayudar a resolver correctamente cada ejercicio si se resolvieron de manera incorrecta. El diseño de dichos mensajes de error fue una parte fundamental en todos los ejercicios de todos los grupos ya que se necesitaba una pequeña guía para los estudiantes pero intentando que la ayuda permitiera el razonamiento del alumno sin darle directamente la respuesta correcta, teniendo en cuenta la particularidad de cada ejercicio y que tipo de ayuda se necesitaba para resolver el mismo.

3.5.1. Ejercicio 1



Este ejercicio marca el inicio de la transición hacia el pensamiento formal en Mileva. A diferencia de los módulos anteriores, se despoja al estudiante de contextos visuales concretos (como luces, puertas o robots) y se introducen nombres (*a, b, c*) para representar las variables. El objetivo es que el sujeto se concentre puramente en la sentencia de asignación y en la naturaleza de la variable como un espacio de memoria.

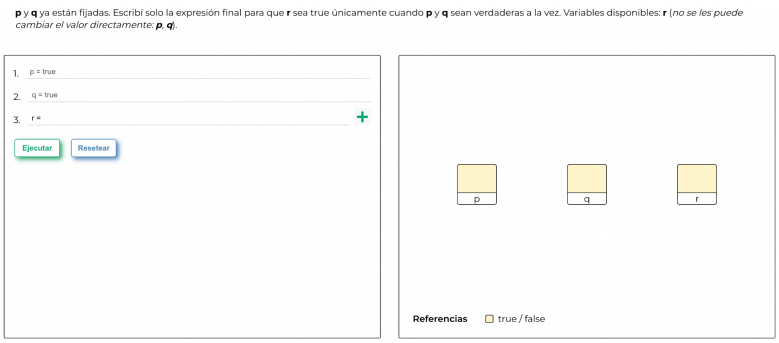
• **Enunciado y Desafío:** El sistema presenta tres cajas etiquetadas como *a*, *b* y *c*. La consigna indica: *"Ajustá los valores de a y b para que a quede true y b false. Usá instrucciones simples del tipo variable = valor y observá cómo el panel refleja el cambio. Variables disponibles: a, b, c."*.

• **Estado Inicial del Código:** El panel presenta una instrucción editable y un espacio para la intervención activa:

1. a = false (Instrucción inicial que el estudiante debe modificar).
2. (Espacio vacío para agregar la instrucción faltante).

• **Resolución:** El estudiante debe editar la primera línea para que asigne true y utilizar el botón + para agregar la instrucción b = false. En el panel de visualización, las variables se representan como cuadrados de colores que muestran su valor interno, permitiendo al usuario verificar el efecto inmediato de cada línea ejecutada.

3.5.2. Ejercicio 2



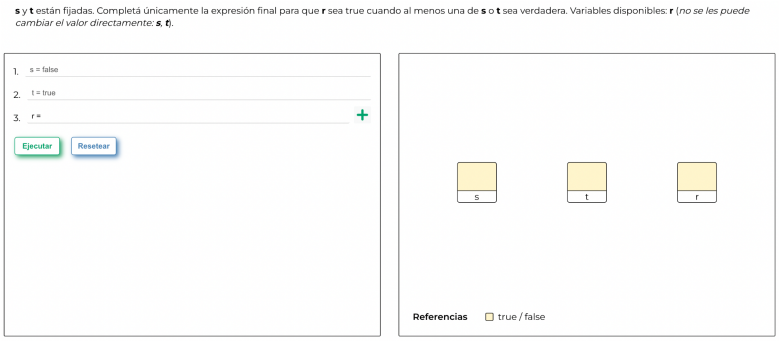
Este ejercicio marca la transición hacia la coordinación lógica abstracta. A diferencia del primer ejercicio de este grupo, donde se realizaban asignaciones directas de valores, aquí el estudiante debe construir una relación de dependencia. El valor de la variable r ya no es una decisión arbitraria del usuario, sino que se define como una función de los estados de otras dos variables, p y q.

- **Enunciado y Desafío:** El sistema presenta tres cajas abstractas (p, q, r). Las variables p y q están fijadas en true y no pueden ser modificadas directamente. La consigna indica: *"p y q ya están fijadas. Escribí solo la expresión final para que r sea true únicamente cuando p y q sean verdaderas a la vez. Variables disponibles: r (no se les puede cambiar el valor directamente: p, q)"*.
- **Estado Inicial del Código:** El panel presenta instrucciones fijas de inicialización y un espacio para la regla lógica:

1. p = true (Instrucción fija).
2. q = true (Instrucción fija).
3. r = (Espacio vacío para completar la expresión).

- **Resolución:** El estudiante debe completar la línea 3 escribiendo r = p AND q o bien r = q AND p. Al ejecutar, Mileva resalta la secuencia en verde. El estudiante observa cómo el autómata *"lee"* los valores de las cajas p y q para determinar el nuevo estado de la caja r.

3.5.3. Ejercicio 3



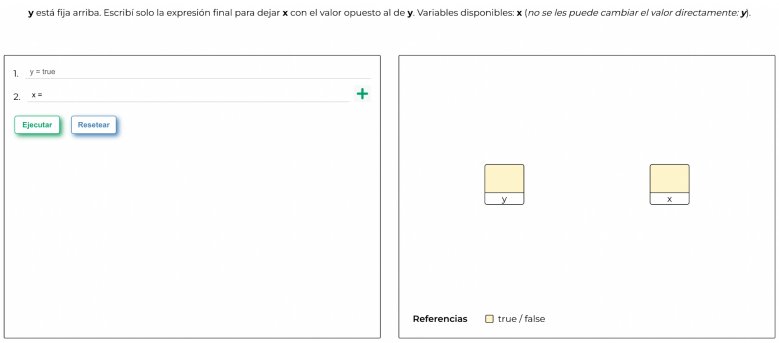
Este ejercicio profundiza en la coordinación lógica mediante la introducción del operador de disyunción (**OR**). El valor de la variable r debe construirse como una respuesta a la presencia de veracidad en al menos uno de los componentes del sistema (s o t), consolidando la idea de que un estado final puede ser alcanzado por múltiples vías.

- **Enunciado y Desafío:** El sistema presenta tres cajas abstractas (s, t, r). Las variables de entrada están fijadas: s en false y t en true. La consigna indica: *"Completá únicamente la expresión final para que r sea true cuando al menos una de s o t sea verdadera"*.
- **Estado Inicial del Código:** El panel presenta una configuración de entrada fija y un espacio para la regla de transformación:

1. s = false (Instrucción fija).
2. t = true (Instrucción fija).
3. r = (Espacio vacío para completar la expresión lógica).

- **Resolución:** El estudiante debe escribir la instrucción r = s OR t o bien r = t OR s. Al presionar "Ejecutar", Mileva resalta la secuencia en verde. El estudiante observa cómo el autómata evalúa los contenidos de las celdas de memoria s y t para determinar que, debido a la veracidad de t, el cuadrado r debe cambiar su estado a true.

3.5.4. Ejercicio 4



Este ejercicio profundiza en la transición hacia el pensamiento formal mediante la aplicación del operador de negación (**NOT**) sobre variables abstractas. A diferencia de los ejercicios de *"práctica de la inversión"* vistos en contextos de robots o luces, aquí se despoja al estudiante de cualquier ayuda figurativa, obligándolo a operar puramente con símbolos y reglas de transformación de memoria.

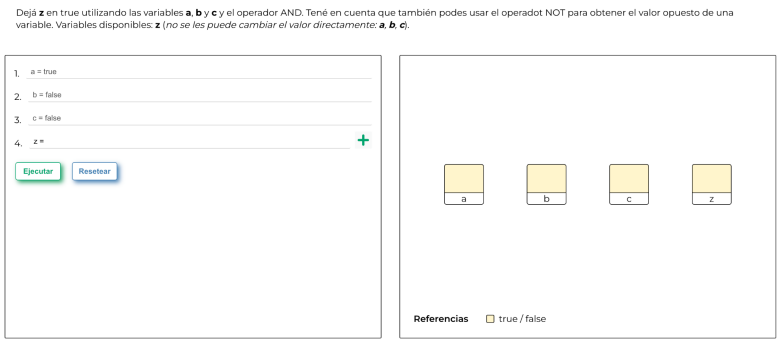
• **Enunciado y Desafío:** El sistema presenta dos cajas abstractas (x, y). La variable y está fijada en true y no puede ser modificada directamente. La consigna indica: *"y está fija arriba. Escribí solo la expresión final para dejar x con el valor opuesto al de y. Variables disponibles: x (No se les puede cambiar el valor directamente: y)"*.

• **Estado Inicial del Código:** El panel presenta una inicialización fija y un espacio para la instrucción del estudiante:

1. y = true (Instrucción fija).
2. x = (Espacio vacío para completar la expresión).

• **Resolución:** El estudiante debe escribir la instrucción x = NOT(y). Al presionar "Ejecutar", Mileva resalta la secuencia en **verde**. El estudiante observa cómo el autómata *"lee"* el valor de la caja y, aplica la inversión lógica y deposita el resultado (false) en la caja x.

3.5.5. Ejercicio 5



A diferencia de los retos anteriores donde se trabajaba con operadores aislados, aquí el estudiante debe realizar una deducción lógica compleja para coordinar el estado de múltiples variables abstractas y alcanzar un valor final específico en una variable de salida.

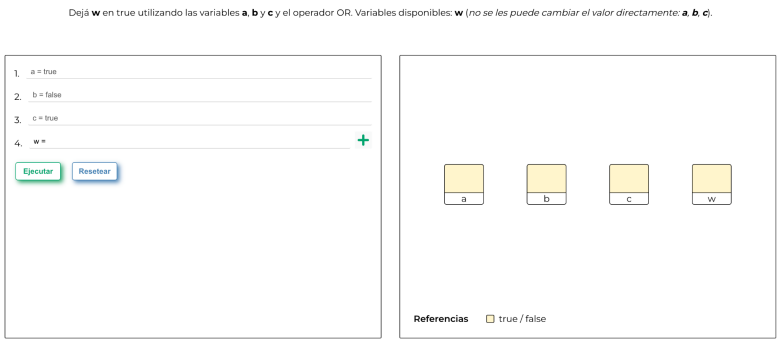
• **Enunciado y Desafío:** El sistema presenta tres cajas abstractas (a,b,c) con valores fijos: a=*true*,b=*false*,c=*false*. La consigna indica: *"Dejá z en true utilizando las variables a, b y c y el operador AND. Tené en cuenta que también podes usar el operador NOT para obtener el valor opuesto de una variable. Variables disponibles: z (no se les puede cambiar el valor directamente: a, b, c)."*.

• **Estado Inicial del Código:** El panel presenta las variables inicializadas y un espacio para la regla de transformación:

1. a = true (Instrucción fija).
2. b = false (Instrucción fija).
3. c = false (Instrucción fija).
4. z = (Espacio vacío para completar la expresión).

• **Resolución:** El estudiante debe construir la expresión z = a AND NOT(b) AND NOT(c) o diferentes ordenamientos de los operandos de dicha expresión. Al ejecutar, Mileva resalta la secuencia en verde. El estudiante observa cómo el autómata *"lee"* los contenidos de las cajas, aplica las inversiones lógicas necesarias y valida la conjunción para transformar el estado visual del cuadrado *z*.

3.5.6. Ejercicio 6



Este ejercicio representa un avance hacia la gestión de sistemas lógicos de múltiples componentes dentro del grupo de formalización. De una manera similar al anterior ejercicio, aquí el estudiante debe construir un esquema de suficiencia para un sistema de tres variables abstractas (a,b,c) con el fin de determinar el estado de una variable de salida (w).

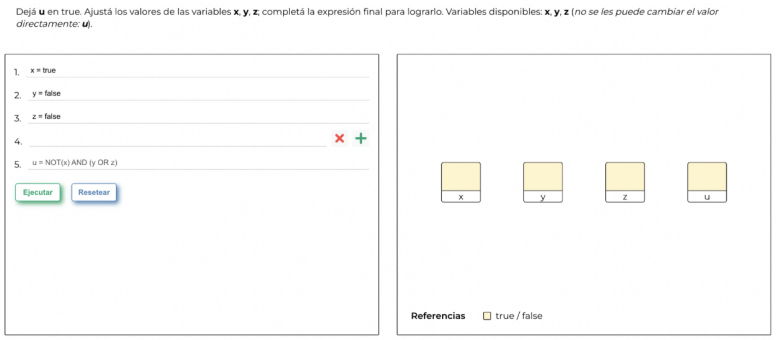
• **Enunciado y Desafío:** El sistema presenta cuatro cajas abstractas (a,b,c,w). Las variables de entrada están fijadas y no pueden modificarse. La consigna indica: *"Dejá w en true utilizando las variables a, b y c y el operador OR"*.

• **Estado Inicial del Código:** El panel presenta las variables inicializadas en un estado mixto (por ejemplo: a=*true*,b=*false*,c=*true*) y un espacio para la regla de transformación:

1. a = true (Fijo).
2. b = false (Fijo).
3. c = true (Fijo).
4. w = (**Espacio vacío para la expresión**).

• **Resolución:** El estudiante debe escribir la instrucción w = a OR b OR c. Al ejecutar, Mileva resalta la secuencia en verde. El sujeto observa cómo el autómata evalúa las tres celdas de memoria y, al encontrar valores verdaderos, transforma el estado visual del cuadrado w a true.

3.5.7. Ejercicio 7



Contrariamente a los retos anteriores que se centraban en operadores aislados o secuenciales, este desafío exige la gestión de una expresión lógica compuesta utilizando símbolos abstractos, integrando la negación (NOT), la conjunción (AND) y la disyunción (OR).

• **Enunciado y Desafío:** El sistema presenta cuatro cajas abstractas (x,y,z,u). El objetivo es dejar la variable u en true. La consigna indica: *"Ajustá los valores de las variables x, y, z; completá la expresión final para lograrlo"*.

• **Expresión Lógica a Evaluar:** u = NOT(x) AND (y OR z).

• **Estado Inicial del Código:** El panel muestra variables de entrada editables inicializadas en un estado de fallo:

1. x = true
2. y = false
3. z = false
4. (Espacio vacío para la intervención activa).
5. u = NOT(x) AND (y OR z) (Instrucción de evaluación fija).

• **Resolución:** El estudiante debe realizar una intervención múltiple en los datos de entrada para satisfacer la jerarquía impuesta por los paréntesis. Debe cambiar x a false y al menos una de las otras dos variables (y o z) a true. Al ejecutar, Mileva resalta la secuencia en verde, permitiendo al sujeto validar su hipótesis sobre cómo se componen los operadores.

Capítulo 4

Arquitectura, diseño e implementación

Este capítulo describe la arquitectura, el diseño y consideraciones relativas a la implementación de la extensión de la herramienta (asignación de variables, formalización y operadores booleanos) y los cambios realizados durante este proyecto en el parser, los componentes y la estructura general de toda la herramienta.

4.1. Arquitectura existente y enfoque de la extensión

4.1.1. Resumen de la arquitectura preexistente

En la primera versión del proyecto (Rosselli y cols., 2023), la herramienta ya estaba definida con una arquitectura clara, documentada en el Capítulo 5 y el Anexo A del documento base.

La arquitectura de Mileva, establecida en su versión inicial, se concibió bajo la premisa de autonomía y simplicidad de despliegue, configurándose como una aplicación web de página única (SPA) ejecutada íntegramente en el cliente. Esta decisión prescinde de la necesidad de un servidor *backend* o una base de datos externa, gestionando tanto el estado de las instrucciones como el progreso del estudiante mediante el almacenamiento local del navegador (*LocalStorage*). El diagrama de despliegue correspondiente y el diagrama de componentes de la versión original se reproducen en el Anexo B (Figuras B.1 y B.2, respectivamente).

Implementada con la librería React, la herramienta se estructura a partir de componentes altamente reutilizables que facilitan la escalabilidad del proyecto y permiten que el entorno de aprendizaje sea accesible desde cualquier dispositivo con conexión a internet sin requerir instalaciones de software adicionales.

Desde el punto de vista funcional, el sistema se organiza en dos áreas críticas: el panel de instrucciones y el panel de visualización. El componente del panel de código (*CodePanel*) actúa como el núcleo de la interfaz de programación, siendo responsable de renderizar el editor de pseudocódigo, gestionar la ejecución secuencial de las sentencias y proporcionar *feedback* visual mediante el resaltado de la instrucción activa. Cada conjunto de ejercicios cuenta con su propio componente de visualización, diseñado específicamente para reflejar los cambios de estado en dominios concretos, como el movimiento de objetos en una recta o el encendido de etiquetas. La orquestación entre estos elementos se delega en un componente principal por cada grupo, el cual interpreta los parámetros de la URL para cargar dinámicamente la configuración del ejercicio desde archivos en formato JSON.

La validación y el procesamiento de las acciones del estudiante residen en dos módulos JavaScript compartidos en la versión original: el *Parser* y el módulo *Arithmetic*. El primero realiza el análisis sintáctico de las entradas del usuario a través de un motor de expresiones regulares, asegurando el cumplimiento estricto de las reglas del lenguaje y el tipado fuerte de las variables; una vez validada la sentencia, *Arithmetic* determina los nuevos estados de memoria, permitiendo que la herramienta sincronice de forma precisa la lógica del programa con la representación visual. El lenguaje simplificado sobre el que opera esta cadena se describe en detalle en el Anexo B (sección B.0.1 y Cuadro B.1).

Este diseño modular garantiza que las reglas de negocio y las restricciones didácticas se apliquen de forma uniforme en todos los ejercicios del Módulo 1, estableciendo la base técnica sólida sobre la cual se integra la extensión de lógica booleana en esta nueva versión de la herramienta Mileva.

4.1.2. Pilares sobre los que se construyó la extensión

La extensión de la herramienta Mileva a la lógica booleana se cimentó sobre los mismos pilares estructurales que definieron la versión original, asegurando la continuidad didáctica y técnica del sistema sin alterar su esencia funcional: la naturaleza SPA, el tipado fuerte, el patrón "componentes por grupo" y los ejercicios definidos en datos.

Sin embargo, la incorporación de expresiones booleanas compuestas (combinaciones arbitrarias de `AND`, `OR`, `NOT`, comparadores relacionales y paréntesis) hizo inviable extender las expresiones regulares del *Parser* original sin comprometer la mantenibilidad del sistema. Por este motivo, la arquitectura se amplió mediante la incorporación de un módulo paralelo, `booleanExpressions.js`, que parsea, valida y evalúa la expresión booleana en un único paso. Ambos componentes (parser.js y booleanExpressions.js) comparten el rol de "regulador didáctico": aseguran que toda instrucción escrita por el estudiante cumpla con las reglas del lenguaje. La motivación técnica de esta decisión y el alcance de la extensión al *Parser* original se desarrollan en detalle en la sección 4.2.4; la Figura B.3 del Anexo B presenta el diagrama de componentes completo.

El *Parser* original recibió una extensión mínima: se incorporó únicamente la forma `var = varA AND varB` entre variables booleanas, requerida por el Gru-

po 1 del Módulo 2 (Variables booleanas). Todos los restantes grupos del Módulo 2 (Expresiones booleanas, Operadores lógicos, Evaluación por circuito corto y Formalización) llaman directamente a `validateBooleanExpressionInput()` de `booleanExpressions.js`, sin pasar por la cadena *Parser* + *Arithmetic*. De este modo se preserva intacto el flujo de procesamiento de los grupos del Módulo 1, mientras que la nueva funcionalidad se aísla en un módulo dedicado, que comparte la responsabilidad con el módulo *Parser* como validador sintáctico. Asimismo, se implementaron restricciones didácticas que bloquean la asignación directa de literales en variables de estado final, obligando al estudiante a construir expresiones lógicas para alcanzar el éxito; estas restricciones se configuran de forma declarativa por ejercicio mediante el conjunto de opciones del nuevo módulo. El Cuadro B.3 del Anexo B sintetiza el contraste entre ambas versiones.

Desde la perspectiva de la interfaz de usuario, se mantuvo el diseño de componentes por grupo, donde cada nuevo conjunto de ejercicios (robot, luces/puertas, circuito corto) posee su propio entorno gráfico pero reutiliza íntegramente el panel de código (*CodePanel*) compartido. Esta decisión de diseño asegura la coherencia de la experiencia de usuario (basada en la ejecución secuencial y el *feedback* inmediato junto a la línea de código) y optimiza el desarrollo al evitar la duplicación de la lógica de edición y validación.

Finalmente, la escalabilidad del sistema se garantizó mediante una arquitectura de ejercicios definidos en datos, conservando la estructura de archivos en el directorio `src/data/exercises/`. Cada nuevo reto de la extensión se define de forma declarativa mediante sus propiedades de estado inicial, objetivos y variables, permitiendo ampliar el catálogo didáctico sin necesidad de modificar el código fuente del núcleo de la aplicación.

Coherentemente con el objetivo de portabilidad, se ratificó el modelo de ejecución exclusiva en el cliente, prescindiendo de servidores *backend* o bases de datos externas. Al persistir el progreso del estudiante únicamente en el almacenamiento local del navegador, Mileva se consolida como una herramienta autónoma, ligera y de fácil despliegue en entornos educativos con conectividad limitada, preservando la simplicidad técnica que ha caracterizado al proyecto desde su inicio.

4.1.3. Criterios para la conservación de la arquitectura original

La decisión de mantener la estructura técnica original de Mileva durante el desarrollo de la extensión respondió a la necesidad de priorizar la incorporación de nuevos contenidos didácticos sobre una base tecnológica ya consolidada y estable.

Dado que el objetivo central de esta extensión consistió en integrar la evaluación de expresiones booleanas y operadores lógicos, se consideró que la arquitectura preexistente ofrecía la flexibilidad necesaria para soportar estas nuevas capacidades sin requerir una reescritura integral del sistema.

Esta estrategia permitió optimizar los esfuerzos en el diseño didáctico de los ejercicios, asegurando que la experiencia de usuario (caracterizada por el panel

de instrucciones, la ejecución secuencial y el feedback inmediato) se mantuviera coherente con el recorrido iniciado en la primera versión de la herramienta.

La estabilidad y la capacidad de reutilización de los módulos compartidos fueron factores determinantes para validar esta continuidad. El diseño modular de la herramienta ya contemplaba una separación clara entre el motor de procesamiento (Parser y Arithmetic) y los componentes de interfaz.

Para los nuevos dominios visuales (robot barrendero, sistemas de puertas controlados por alarma, etc.) se desarrollaron los componentes de visualización correspondientes.

Al integrar estos elementos dentro del modelo de "componentes por grupo", se garantizó la integridad del sistema y se facilitó la gestión de los nuevos retos sin alterar el flujo de ejecución secuencial que caracteriza a todos los ejercicios.

En síntesis, la extensión de Mileva se consolidó sobre sus pilares originales, ampliando el lenguaje y los datos para dar soporte a la lógica formal mediante un módulo paralelo dedicado, mientras conservaba la autonomía y ligereza técnica que definen al proyecto.

4.1.4. Migración a Vite como Build Tool

Se implementó una migración tecnológica hacia la build tool Vite reemplazando la actual, create react app la cual fue deprecada en 2023. Este cambio se orientó exclusivamente a modernizar el entorno de desarrollo y optimizar los tiempos de construcción para producción, sin alterar el modelo de ejecución descentralizado.

La nueva configuración optimiza el despliegue y facilita la integración de herramientas de testeo modernas como Vitest, asegurando que la herramienta siga siendo una aplicación de página única (SPA) ligera, autónoma y fácil de desplegar en entornos educativos sin dependencia de servidores backend.

En resumen, estas actualizaciones de infraestructura no solo mejoran el rendimiento técnico de la aplicación, sino que proporcionan un entorno de desarrollo más robusto para la incorporación de futuras extensiones didácticas.

El detalle del proceso de migración se presenta en la sección B.0.5 del Anexo B.

4.2. Diseño de los ejercicios

4.2.1. Visualización

La interfaz de usuario de Mileva se estructura bajo los principios del Sistema de Interacción Híbrida (HIS), facilitando un entorno donde la transparencia del artefacto digital permite al estudiante validar sus respuestas de forma inmediata.

La pantalla se organiza en dos áreas funcionales críticas: a la izquierda se sitúa el panel de instrucciones o código (CodePanel), encargado de la edición y ejecución secuencial del pseudocódigo; a la derecha se despliega el panel de

visualización, cuyo contenido gráfico se adapta dinámicamente según el dominio del grupo de ejercicios seleccionado.

Esta disposición asegura que el sujeto mantenga un foco de atención dual, relacionando la sintaxis formal con la transformación física de los objetos digitales.

Para la extensión, las representaciones visuales han sido diseñadas para acompañar la transición del pensamiento del estudiante desde la etapa Intra hasta la etapa Trans, en el marco de la tríada de etapas de la Epistemología Genética.

En el Grupo 1, el panel presenta grillas de bombitas o conjuntos de puertas cuyos estados de encendido o apertura se vinculan directamente a variables booleanas específicas; en el Grupo 2, se incorporan leyendas e indicadores de color que explicitan los tipos de datos (booleano, numérico o texto) para guiar la construcción de comparaciones.

Por su parte, el Grupo 3 integra la figura del robot barrendero, el cual reacciona de forma reactiva a operadores lógicos complejos (AND, OR, NOT), permitiendo que el estudiante observe cómo la coordinación de múltiples condiciones de entrada gobierna el comportamiento global del sistema.

En los niveles de mayor abstracción, la visualización se desprende de lo figurativo para centrarse en los procesos internos de la máquina. En el Grupo 4 (circuito corto), el panel de visualización se vuelve "transparente" al resaltar en amarillo la subexpresión que el autómata está evaluando en cada instante y utilizar carteles naranjas para notificar la detección de evaluaciones perezosas o por circuito corto.

Finalmente, en el Grupo 5 (formalización), se prescinde de analogías físicas y se presentan cajas abstractas que representan celdas de memoria, induciendo al estudiante a razonar puramente sobre el flujo de datos y la doble semántica de la asignación.

Un elemento didáctico fundamental de este diseño es la gestión del tiempo y el feedback secuencial. Al presionar "Ejecutar", el sistema no entrega un resultado instantáneo, sino que introduce un retraso controlado entre instrucciones, resaltando en verde la línea activa.

Esta decisión técnica permite que el estudiante perciba el cambio de estado de cada variable como un evento discreto, favoreciendo la toma de conciencia sobre la secuencialidad y el orden de los operandos.

Asimismo, los mensajes de error se despliegan de forma embebida junto a la instrucción fallida, proporcionando pistas semánticas y de tipado que inducen a la abstracción reflexiva y a la corrección autónoma sin requerir la intervención constante de un docente.

4.2.2. Lenguaje de las sentencias (extensiones para expresiones booleanas)

El lenguaje de programación simplificado que utiliza Mileva conserva su naturaleza de tipado fuerte, asegurando que cada variable actúe como un contenedor con una categoría de dato inmutable (numérico, booleano o texto) que

no puede ser alterada durante la ejecución. Para la extensión se mantienen las reglas estructurales de la versión original (Cuadro B.1 del Anexo B) y se incorporan nuevas formas para soportar la evaluación de expresiones booleanas, sintetizadas en el Cuadro B.2 del Anexo B.

La principal extensión técnica de esta fase consiste en el soporte de expresiones booleanas compuestas, formadas por los operadores `AND`, `OR`, `NOT`, los comparadores relacionales (`>`, `<`, `>=`, `<=`, `==`, `!=`) y paréntesis.

El tipado se adapta dinámicamente según el contexto del grupo de ejercicios: mientras que en el Grupo 5 (formalización) las variables operan como abstracciones lógicas puras identificadas con letras, en los grupos de control reactivo (1, 2 y 3) los tipos se vinculan a estados del dominio físico, como el encendido de una bombita o la temperatura de un sensor.

Un pilar fundamental del diseño lingüístico en esta etapa es la implementación de reglas didácticas restrictivas para las variables que representan estados finales o decisiones del sistema. Para evitar que el estudiante recurra a soluciones instrumentales o superficiales (como asignar directamente un valor de éxito), el sistema bloquea activamente la asignación de literales (`true`/`false`) cuando la consigna exige explícitamente la construcción de una relación lógica. Estas restricciones se configuran de forma declarativa por ejercicio mediante el conjunto de opciones del módulo `booleanExpressions.js` (por ejemplo, `exactBinaryForLhs`, `requireVarsInExprForLhs` o

`disallowDirectBooleanLiteralFor`), sin necesidad de modificar la lógica de validación. Ante estos intentos, la herramienta devuelve mensajes de retroalimentación que orientan al sujeto hacia el uso correcto de los operadores lógicos requeridos.

Finalmente, el catálogo de mensajes de error ha sido expandido para integrar la semántica booleana dentro de las respuestas provistas por la herramienta. Además de detectar fallos comunes como la asignación invertida o tipos incompatibles, el sistema proporciona alertas específicas ante el uso inválido de operadores, como el intento de aplicar un `AND` sobre datos no booleanos. Estos mensajes se despliegan de forma embebida junto a la instrucción fallida, permitiendo que el estudiante identifique inconsistencias en tiempo real y reflexione sobre su propio código, transformando así el error en un motor fundamental para la construcción del conocimiento computacional.

4.2.3. Componentes

Cada grupo de ejercicios de la extensión (Grupos 1 a 5 del Módulo 2) cuenta con un componente React que orquesta el panel de código, la visualización propia del grupo y la lógica de estado. Dicho componente delega la validación de instrucciones en un módulo de lógica del grupo. La diferencia clave respecto de la versión original radica en a qué motor de evaluación delega cada grupo:

- El **Grupo 1 del Módulo 2 (Variables booleanas)** continúa utilizando la cadena `parseInput()` → `calculateNewValue()`, ya que sus sentencias

mantienen una estructura simple (asignación de literal, copia entre variables y la forma `var = varA AND varB`) reconocida en el *Parser* original mediante una extensión puntual.

- Los **restantes grupos del Módulo 2** (Expresiones booleanas, Operadores lógicos, Evaluación por circuito corto y Formalización) invocan directamente a `validateBooleanExpressionInput()` del módulo

  `booleanExpressions.js`, que parsea, valida y evalúa la expresión booleana arbitraria en un único paso.

La Figura B.3 del Anexo B muestra el diagrama completo de componentes y la sección B.0.4 describe el funcionamiento interno de `booleanExpressions.js`.

La identificación del ejercicio activo se obtiene desde la URL (por ejemplo, `/modulo/2/grupo/5/ejercicio/3`). A partir de ella se cargan la definición del ejercicio desde su archivo de datos y el componente de visualización correspondiente al grupo.

4.2.4. Parser y validación de expresiones booleanas

El módulo *Parser* (`src/utils/parser.js`) constituye el núcleo de procesamiento del Módulo 1 de Mileva y continúa siendo responsable de analizar cada sentencia ingresada en esos ejercicios para determinar su validez sintáctica y semántica. El flujo de validación se apoya en un sistema de expresiones regulares y evaluaciones condicionales que aseguran que el código cumpla estrictamente con las reglas del lenguaje y el tipado fuerte definido para cada ejercicio. El proceso comienza verificando que la entrada represente una asignación válida mediante la presencia del operador `=`; en esta instancia, el sistema identifica si el nombre a la izquierda corresponde a una variable permitida, detectando errores comunes como la omisión de valores o la asignación invertida (`valor = variable`), en cuyo caso devuelve mensajes de orientación inmediata.

En esta extensión, el *Parser* fue ampliado de forma puntual para reconocer una única forma adicional: `var = varA AND varB` entre variables booleanas, requerida por el Grupo 1 del Módulo 2 (Variables booleanas). Esta validación exige que los tres operandos involucrados sean de tipo booleano; de lo contrario, el sistema intercepta la ejecución y notifica que las operaciones de conjunción están restringidas exclusivamente a este tipo de datos.

A partir del Grupo 2 del Módulo 2 (Expresiones booleanas), los ejercicios requieren que el estudiante construya expresiones lógicas arbitrariamente compuestas, con combinaciones de `AND`, `OR`, `NOT`, comparadores relacionales (`>`, `<`, `>=`, `<=`, `==`, `!=`) y paréntesis. Extender las expresiones regulares del *Parser* original para cubrir este espacio hubiera implicado una explosión combinatoria de patrones y lógica de precedencia difícil de mantener, por lo que la validación y evaluación de estas expresiones se delegó al módulo `booleanExpressions.js`, descrito en detalle en la sección B.0.4 del Anexo B. Este módulo (booleanExpressions.js) opera como una cadena de evaluación paralela e independiente del

Parser pero ambos cumplen el mismo rol, son validadores sintácticos. La manera de validar expresiones de un módulo y otro es distinta (se explica en B.0.4) pero trabajan en conjunto con el mismo objetivo, extender el dominio de las expresiones admitidas.

4.3. Implementación

4.3.1. Ejercicios en formato JSON (extensión)

La definición y organización de los conjuntos de ejercicios que integran esta extensión de la herramienta se estructuran mediante archivos JavaScript que exportan objetos en formato JSON, los cuales se encuentran indexados por el número de ejercicio correspondiente.

Esta arquitectura de datos permite que la herramienta sea altamente extensible, facilitando la adición o modificación de retos didácticos sin necesidad de alterar la lógica central del parser o de los componentes de visualización.

Si bien la estructura interna de cada objeto JSON varía para adaptarse a la semántica específica de cada grupo, todos los ejercicios comparten propiedades fundamentales, como la descripción, que contiene el texto instructivo desplegado en la parte superior de la interfaz, y el arreglo "givenInstructions", que define las sentencias precargadas, diferenciando entre aquellas que son fijas y las que permiten la edición por parte del estudiante.

En el caso de los grupos con un fuerte componente visual y físico, como el Grupo 1 (luces y puertas) y el Grupo 3 (robot), los objetos JSON incorporan propiedades adicionales para modelar el entorno digital. Entre ellas se destacan el "initialState" y el "goal", que definen la configuración o estado inicial y el estado objetivo del sistema, así como un identificador de visualización que determina qué componente gráfico debe renderizarse (por ejemplo, "lights" o "doors").

Por el contrario, el Grupo 5 (formalización) prescinde de un dominio visual concreto y centra su estructura de datos en la definición de variables abstractas, sus tipos y los "finalValues" o condiciones lógicas que determinan el éxito de la tarea basándose estrictamente en el estado de la memoria interna.

La implementación técnica de estos conjuntos se encuentra centralizada en el directorio src/data/exercises/, donde residen archivos específicos como formalizationModule2.js, operatorsGroup.js y shortCircuitGroup.js.

Esta segmentación permite que la herramienta cargue dinámicamente la lógica del ejercicio basándose en los parámetros de la URL, asegurando que las reglas didácticas y las restricciones de cada reto se apliquen de forma uniforme a través de los módulos compartidos de la aplicación

4.3.2. Página de inicio y navegación (extensión)

La puerta de entrada al nuevo contenido didáctico es una página de inicio rediseñada que presenta de forma clara los módulos disponibles y, específicamente

dentro de la extensión desarrollada, los cinco grupos de ejercicios que componen la extensión a booleanos.

Para fomentar la autonomía del estudiante, el sistema implementa un mecanismo de persistencia basado en el almacenamiento local del navegador (LocalStorage), lo que permite registrar y visualizar el progreso mediante indicadores visuales (ticks verdes) que se mantienen incluso tras cerrar la sesión de navegación.

El flujo entre las diferentes actividades se gestiona a través de un sistema de rutas parametrizadas, el cual interpreta el módulo, el grupo y el número de ejercicio directamente desde la dirección URL para orquestar la carga de datos.

Una característica relevante de este diseño es la capacidad de gestionar ejercicios ocultos mediante la propiedad "hidden" en el JSON; esto permite que el sistema de navegación redirija automáticamente al usuario hacia el ejercicio visible más cercano dentro de un mismo grupo, evitando interrupciones en la secuencia didáctica planeada.

4.3.3. Tutorial y uso de la herramienta

Para minimizar la carga cognitiva asociada al manejo de la interfaz y permitir que el estudiante se concentre exclusivamente en los obstáculos conceptuales, la herramienta incluye un tutorial inicial que demuestra las funcionalidades básicas del sistema.

A través de una secuencia de pasos guiados, se explica el propósito del panel de instrucciones (CodePanel) y la mecánica del botón "Ejecutar", resaltando visualmente la línea de código activa para que el sujeto comprenda la naturaleza secuencial del autómata.

Este comportamiento tutorial es transversal a todos los grupos de la extensión de Mileva, adaptando únicamente el contenido de la ventana de visualización según el contexto del ejercicio en curso.

De esta manera, ya sea interactuando con robots, luces o variables abstractas, el estudiante experimenta un entorno de ejecución predecible y coherente que facilita el ciclo de interacción recíproca.

Capítulo 5

Pruebas

Una vez finalizada la implementación de los ejercicios de cada grupo, se llevaron a cabo dos instancias de pruebas con el objetivo de obtener datos primarios sobre el funcionamiento y la efectividad de la herramienta desde el punto de vista del uso por parte de distintos usuarios. Esos datos nos permitieron realizar ajustes como se describe en este capítulo.

Se realizaron dos iteraciones con distintos objetivos, en la primer iteración se buscó recibir feedback y validar la herramienta con los distintos participantes. En la segunda iteración se buscó validar los cambios realizados a partir del feedback recibido con nuevos participantes y comparar en términos porcentuales para ver si hubo una mejora en la cantidad de ejercicios resueltos por parte de los estudiantes.

El foco principal de este estudio estuvo centrado en el **Módulo 2**, ya que el proyecto de grado se orienta específicamente a la implementación de este módulo y a analizar la adopción de los alumnos frente a esta nueva propuesta dentro de la herramienta Mileva. En este sentido, las pruebas buscaron evaluar si los ejercicios eran resolubles e indagar si los estudiantes lograban comprender los conceptos introducidos, por medio de la observación del comportamiento de los estudiantes.

Asimismo, se buscó validar si los errores esperados en el diseño de los ejercicios efectivamente se producían, y si las visualizaciones y mensajes de error embebidos eran suficientes para permitir la corrección autónoma sin intervención externa. La literatura previa documenta dificultades persistentes que tienen los estudiantes principiantes al razonar con lógica proposicional y al traducir reglas del lenguaje natural a expresiones booleanas (Herman, Loui, Kaczmarczyk, y Zilles, 2012), así como concepciones erróneas en torno a variables y lógica booleana en entornos de programación por bloques (Grover y Basu, 2017), aspectos que se buscaron observar también en este estudio.

5.1. Enfoque del estudio

Si bien el interés principal del estudio se centró en el Módulo 2 (extensión de la herramienta), se decidió incluir el Módulo 1 como instancia previa para los participantes. Esto se debe a que la mayoría de los estudiantes no contaban con experiencia previa en programación, por lo que era necesario proporcionar un contexto mínimo sobre el funcionamiento de la herramienta.

El Módulo 1 permitió introducir conceptos básicos como la asignación de variables, el flujo de ejecución y la interpretación de resultados a partir de la visualización. De esta forma, actuó como un introductor que facilitó la comprensión de los ejercicios del Módulo 2.

No obstante, a partir de la primera iteración de pruebas, se observó que no era necesario completar la totalidad de los ejercicios del Módulo 1. En particular, aquellos ejercicios relacionados con la asignación de variables y el concepto de estado resultaron suficientes para que los participantes comprendieran la dinámica de la herramienta.

En consecuencia, en las instancias posteriores se ajustó el protocolo, reduciendo el Módulo 1 a un subconjunto de ejercicios clave, optimizando así el tiempo de las pruebas sin afectar la calidad del análisis del Módulo 2.

5.2. Participantes

En total participaron 15 personas en las 2 iteraciones de pruebas entre 15 y 26 años, las cuales no tenían un conocimiento previo de programación.

En la primera instancia participaron 8 estudiantes y se realizó principalmente con un grupo de estudiantes pertenecientes a la categoría sub-20 de un club deportivo. La participación se dio en el marco de una actividad voluntaria facilitada por uno de los autores del proyecto, quien se desempeñaba como colaborador en dicho contexto.

Este grupo resultó particularmente relevante, ya que representa el público objetivo de la herramienta: jóvenes sin experiencia previa en programación.

En la segunda instancia particiaron 7 personas entre 15 y 26 años de edad (distintas a las primeras 8). Dicha instancia tuvo el objetivo de validar los cambios realizados luego de recibir el feedback de la primer iteración. En esta segunda etapa también se hicieron ciertas modificaciones a los ejercicios a medida que íbamos recibiendo feedback de los distintos voluntarios.

Desde el punto de vista metodológico, esta selección de participantes entre 15 y 26 años permite observar procesos de construcción de conocimiento en sujetos que no poseen esquemas previos formalizados en programación, lo que resulta especialmente valioso para el análisis didáctico.

5.3. Metodología

Las pruebas se realizaron de manera individual, donde nosotros actuamos como facilitadores para los estudiantes encargandonos de observar y registrar el desempeño de cada participante sin intervenir directamente en la resolución.

Cada sesión incluyó:

- Resolución de ejercicios (Módulo 1 reducido + Módulo 2 completo)
- Registro de errores y estrategias utilizadas
- Aplicación de ayudas escalonadas en caso de bloqueo
- Preguntas finales de comprensión

El protocolo de ayuda se estructuró en tres niveles:

- A1: reinterpretación de la consigna
- A2: observación de la visualización
- A3: guía sobre el formato de la solución

Esto permitió evaluar en qué medida la herramienta era suficiente para guiar al estudiante sin asistencia externa.

5.4. Resultados

Los resultados se organizan en función de los distintos grupos de ejercicios que componen el Módulo 2 de MILEVA.

En términos generales, se observa que a medida que aumenta el nivel de abstracción requerido por los ejercicios, disminuye la capacidad de resolución autónoma y aumenta la dependencia de ayudas externas, lo cual es un resultado esperado.

5.4.1. Grupo 1 — Variables booleanas

En este primer grupo, los participantes lograron, en su mayoría, resolver los ejercicios sobre variables booleanas y su representación mediante los valores true y false. Sin embargo, algunos participantes tuvieron problemas al principio para entender qué representa cada valor en la práctica: por ejemplo, no tenían claro que true equivale a “encendido”.

Esta confusión no se debió a una falta de comprensión lógica, sino a una falta de correspondencia inmediata entre el valor abstracto y su representación concreta. En este sentido, la visualización cumplió un rol fundamental como mediadora del aprendizaje, permitiendo que los estudiantes ajustaran progresivamente su interpretación a partir de la observación del comportamiento del sistema.

En los últimos ejercicios de este grupo, los que utilizan la alarma, algunos participantes presentaron dudas sobre la condición de la alarma, lo que sugiere que la interpretación de reglas lógicas externas puede generar cierta confusión inicial, lo cual está también dentro de lo esperado.

En el último ejercicio, el del telón que "tapa" el estado inicial, muchos de los participantes demostraron un pensamiento estratégico al decidir cerrar una puerta inicialmente "por las dudas"para asegurar un estado seguro antes de proceder, lo que muestra una toma de conciencia sobre el control de estados.

5.4.2. Grupo 2 — Expresiones booleanas

En el segundo grupo se introduce la necesidad de construir expresiones a partir de relaciones entre variables, lo que implica un cambio significativo respecto al grupo anterior.

Uno de los errores más frecuentes observados fue la tendencia a asignar valores booleanos directos (por ejemplo, r = true), en lugar de construir una expresión lógica que produzca dicho resultado. Este comportamiento evidencia que la sentencia de asignación donde las expresiones no son numéricas, aún cuando los operadores son los relacionales, representa un desafío importante.

A pesar de ello, en muchos casos la combinación de visualización y mensajes online permitió corregir este tipo de errores sin intervención externa, lo que sugiere la efectividad de la herramienta para orientar al estudiante hacia la construcción correcta de la solución.

5.4.3. Grupo 3 — Operadores lógicos (AND, OR, NOT)

El tercer grupo constituye el núcleo conceptual del Módulo 2 y, en consecuencia, el punto donde se concentraron muchas dificultades.

En relación al operador AND, varios participantes interpretaron la condición como opcional, es decir, asumieron que bastaba con una condición verdadera para que el resultado fuera positivo. Este error refleja una comprensión parcial de la lógica de conjunción, donde no se logra internalizar la necesidad de simultaneidad, comportamiento consistente con las concepciones erróneas reportadas en la literatura sobre lógica booleana en estudiantes principiantes (Herman y cols., 2012; Grover y Basu, 2017).

En el caso del operador NOT, se observaron dificultades aún más marcadas. Muchos participantes intentaron modificar directamente el valor final de la variable, en lugar de aplicar la operación de inversión sobre otra variable. Este comportamiento sugiere una falta de comprensión de la operación como transformación lógica, y no como asignación arbitraria de valor (Herman y cols., 2012).

Asimismo, los ejercicios que requieren la combinación de múltiples operadores evidenciaron dificultades en la construcción de expresiones completas. En estos casos, los estudiantes tendieron a fragmentar el problema o a recurrir a estrategias de ensayo-error, en lugar de planificar la solución de manera estructurada.

Un hallazgo particularmente relevante fue el impacto de la visualización en este grupo. Cuando la representación visual del sistema era clara y consistente, los estudiantes lograban avanzar con mayor autonomía. Sin embargo, en aquellos casos donde existían inconsistencias (por ejemplo, estados visuales que no coincidían con los valores lógicos los cuales fueron arreglados luego de la primer iteración), se generaban bloqueos incluso en participantes con buen desempeño.

En ocasiones se observó que algunos participantes ingresaban instrucciones de la forma (robotEncendido = true AND ordenBarrer = true) en las cuales se evidencia que intentaban escribir la lógica completa en la asignación en lugar de inicializar previamente las variables para que la expresión evalúe a true. Por otro lado, también es posible que dichos participantes hayan confundido el operador de comparación de igualdad con el de asignación, a pesar de que en ejercicios previos se marque claramente esta diferencia.

También se logró evidenciar la utilización del concepto de circuito corto de forma inconsciente al cargar los valores, lo que sugiere que la herramienta permite descubrimientos intuitivos antes de la propia introducción del tema (circuito corto se da en el siguiente grupo de ejercicios). Se describen dos casos a continuación.

Los siguientes ejemplos fueron reconstruidos a partir de las observaciones registradas durante las sesiones con los participantes. Se seleccionaron dos casos representativos del comportamiento descrito previamente en esta sección.

Caso 1 — Ejercicio 5 (operador OR)

Durante la resolución del Ejercicio 5, un participante debía completar las instrucciones para que `robotEncendido = botonEncendido OR encendidoPorVoz` evaluara a `true`. El estado inicial del código presentaba ambas variables en `false`.

El participante agregó únicamente la instrucción `botonEncendido = true`, ejecutó el código y, al ver que el robot se activaba, dio el ejercicio por resuelto sin modificar la variable `encendidoPorVoz`. Al consultarle el facilitador qué valor tenía `encendidoPorVoz` en ese momento, el participante respondió que no importaba porque el robot "ya había arrancado con el botón". Esta respuesta evidencia una comprensión intuitiva de la suficiencia lógica del OR: una vez que el primer operando es verdadero, el segundo deja de ser relevante para el resultado, lo cual es precisamente el comportamiento del circuito corto, aunque el participante no lo reconociera como tal en ese momento.

Caso 2 — Ejercicio 10 (operador OR)

Durante la resolución del Ejercicio 10, un participante debía completar las instrucciones para que `robotSeApaga = (bateria == 0 OR pisoLimpio OR NOT(ordenBarrer))` evaluara a `true`. El estado inicial presentaba `bateria = 2`, `pisoLimpio = false` y `ordenBarrer = true`.

El participante modificó únicamente `bateria = 0` y ejecutó el código, logrando que el robot se apagara. A continuación, sin que nadie se lo indicara,

comenzó a probar qué ocurría si devolvía `bateria` a su valor original mientras dejaba las otras variables sin modificar. Al no obtener el apagado, intentó cambiar `pisoLimpio = true` y ejecutó nuevamente, obteniendo el resultado esperado. En ese punto el participante comentó: *"con cualquiera de las dos alcanza"*.

Lo llamativo fue que en ningún momento intentó modificar `ordenBarrer`, a pesar de que era la tercera condición disponible. Este comportamiento sugiere que el participante había construido intuitivamente la noción de suficiencia lógica del OR: encontrada una condición verdadera, las restantes dejan de ser relevantes para el resultado, anticipando el mecanismo del circuito corto antes de su introducción formal.

Finalmente, también se identificaron dificultades en la interpretación del objetivo final del ejercicio. En algunos casos, los estudiantes no lograban identificar claramente si el objetivo era encender, apagar o modificar el estado del sistema, lo que dificultaba la validación de sus propias soluciones.

5.4.4. Grupo 4 — Evaluación de expresiones (circuito corto)

En el cuarto grupo se introduce el concepto de evaluación de expresiones, incluyendo el comportamiento de circuito corto en operadores lógicos.

Cabe destacar que la evaluación por circuito corto no es un concepto completamente ajeno a la experiencia cotidiana. En el lenguaje natural aplicamos una lógica similar sin darnos cuenta: si alguien piensa "voy al cine si tengo tiempo y dinero", en el momento en que sabe que no tiene tiempo, directamente descarta la condición sin preguntarse si tiene dinero o no. El segundo factor deja de importar.

Sin embargo, esta familiaridad puede ser también una fuente de confusión. Por un lado, el lenguaje natural no siempre es preciso ni consistente en cómo expresa las condiciones. Por ejemplo, en matemática estamos acostumbrados a expresiones como $0 < x < 5$, donde todos los términos se evalúan de forma completa y explícita. En programación, en cambio, la evaluación puede detenerse antes de analizar todos los operandos. Esta diferencia entre la intuición cotidiana y el comportamiento formal puede dificultar que los estudiantes reconozcan cuándo y por qué parte de una expresión no llega a evaluarse.

En este contexto, se observó un patrón de resolución recurrente: varios de los participantes tendían a asignar todos los valores en true con el objetivo de garantizar que la condición se cumpliera. Esta estrategia, si bien permite alcanzar el resultado esperado en algunos casos, no refleja una comprensión real de lo que se estaba evaluando.

Este comportamiento puede interpretarse como una forma de resolución instrumental, donde el estudiante prioriza el resultado por sobre la comprensión del mecanismo subyacente.

Por otro lado, otros participantes sí notaron correctamente en los ejercicios iniciales que el segundo operando del OR nunca se resaltaba en amarillo cuando el primero valia true, asi como en el caso del AND cuando el primer

operando vale false, validando que la herramienta logra mostrar la omisión por optimización.

También se pudo evidenciar que aunque hicieran uso del circuito corto, en ocasiones, por ejemplo en los ejercicios de circuito corto del operador OR, intentaban inicializar el primer operando en true (lo cual es correcto), pero también inicializaban en false el/los siguiente/s operando/s para "forzar" el circuito corto, sin lograr comprender que el valor de dichos operandos no importan ya que nunca se llegan a evaluar.

En conclusión, si bien en algunos casos la herramienta ayudó en la comprensión a nivel básico del concepto de circuito corto, se evidenció que, en base a las dificultades presentadas por los participantes, quizás la visualización no lograba representar de manera suficientemente clara el proceso de evaluación paso a paso. En particular, varios de los participantes no percibían que ciertas partes de la expresión no eran evaluadas debido al corto circuito, lo que limita la construcción de una comprensión profunda del concepto. En este caso tampoco podemos ignorar que el concepto de circuito corto no resulta un tema sencillo de asimilar, menos aún para estudiantes sin conocimientos previos en programación.

5.4.5. Grupo 5 — Formalización

El último grupo introduce un nivel de abstracción mayor, donde se requiere construir expresiones lógicas sin apoyo directo de la visualización concreta.

En este contexto, se observó una mayor dependencia de las pistas proporcionadas por la herramienta. Muchos participantes lograban resolver los ejercicios únicamente cuando contaban con ejemplos o sugerencias de formato, lo que indica que la construcción autónoma de expresiones aún no está consolidada.

Además, se identificó una dificultad en la planificación de la solución, ya que algunos estudiantes intentaban resolver el problema mediante múltiples intentos, en lugar de estructurar una expresión completa desde el inicio.

En el último ejercicio de este grupo se notó que se cometieron algunos errores de sintaxis al intentar combinar AND y NOT (b AND c NOT a), logrando el éxito recién tras varios intentos y ayuda externa para estructurar la expresión como a AND (NOT b AND NOT c) o bien a AND (NOT b) AND (NOT c). Esto lleva a pensar que se podría reforzar más sobre la precedencia de los operadores y el uso de los paréntesis, ya que si bien son temas que son introducidos levemente en algunos ejercicios previos, a los participantes no les resulta natural y suelen en ocasiones apoyarse en agregar paréntesis que no son necesarios.

5.5. Cambios implementados

En la primera iteración se dejó entrever que a los estudiantes se les dificultaba saber qué poner en las instrucciones, en el primer grupo les era complejo por ejemplo saber que tenían que escribir específicamente los valores true or false. Debido a esto se agregaron una serie de ejercicios de introducción que sirvieron para ayudar a que los estudiantes que participaron que en la segunda iteración

superaran esa dificultad. Dichos ejercicios fueron ubicados estratégicamente al inicio de cada grupo y antes de la resolución de lo que nosotros denominamos como ejercicios complejos. La primera iteración con estudiantes nos permitió discernir qué ejercicios necesitaban estos ejercicios de introducción, donde el estudiante solo presiona ejecutar y puede comprobar qué sucede cuando se ejecutan ciertas instrucciones con ciertos valores. Ese input fue fundamental para una correcta resolución por parte de los estudiantes en la segunda iteración.

Otra de las conclusiones que se obtuvieron de la primera iteración de ejercicios fue que las pruebas eran extensas y se volvían tediosas para los estudiantes. En primera instancia les pedimos a los estudiantes que realizarán todos los ejercicios de Mileva 1 para poder tener contexto de cómo afrontar los ejercicios de Mileva 2, pero el problema con esto es que las pruebas llegaron a demorar 2 horas y cuando llegaba la hora de resolver los ejercicios de Mileva 2 que eran los importantes para nosotros y de los que necesitábamos feedback el estudiante se encontraba ya agotado mentalmente. Para resolver esto, en la segunda iteración se les pidió a los estudiantes que resolvieran solo algunos ejercicios de Mileva 1 y no todos, les pedimos que resolvieran solamente los que nosotros creímos fundamentales y necesarios para que pudieran comprender el contexto de Mileva y resolver los ejercicios de Mileva 2 de una buena manera.

Por último nos dimos cuenta de que habían muchos ejercicios que eran repetitivos en Mileva 2, a modo de ejemplo en el grupo 1 teníamos ejercicios donde teníamos que prender 3 bombitas de luz y luego 4, abrir 2 puertas, luego 3 y luego 4 puertas. Estos ejercicios si bien eran interesantes para que el estudiante pudiera experimentar variaciones que le permitieran construir nuevos conocimientos, hacían que las pruebas se alargasen y que cuando tocaba un nuevo grupo el estudiante estuviera agobiado al resolver tantos ejercicios. Lo que hicimos fue ocultar esos ejercicios repetitivos y logramos en la siguiente iteración con alumnos una mayor fluidez a lo largo de la resolución de los ejercicios.Dichos ejercicios que ocultamos se encuentran en el anexo de este informe.

Aparte de los cambios mencionados que fueron los más importantes, se introdujeron las siguientes mejoras a Mileva 2 en base a los feedbacks obtenidos en las distintas iteraciones:

- Refinamiento de mensajes de error y en la propuesta del ejercicio (mayor especificidad y orientación a la acción)
- Se añade un listado de variables disponibles para guiar al estudiante hacia la correcta resolución
- Se agrega también una lista de variables no modificables directamente
- Ajuste de estados iniciales para mejorar la percepción de causa-efecto
- Corrección de inconsistencias entre visualización y lógica
- Mejora en la representación del flujo de evaluación
- Incorporación de elementos visuales dinámicos (animaciones)

5.6. Síntesis de los resultados

En este capítulo se presentan los resultados de las iteraciones con los estudiantes, antes y después de las mejoras realizadas. Cabe destacar que las mejoras surgieron producto de la primera iteración, por lo tanto se puede apreciar una considerable mejora entre la primera y la segunda iteración.

Nosotros, como facilitadores cumplimos un rol de observación durante las sesiones, interviniendo únicamente como último recurso cuando el estudiante no lograba avanzar por sus propios medios ni con la ayuda que ofrece la plataforma.

Las tablas muestran los resultados para cada grupo de ejercicios, en la primera iteración participaron 8 estudiantes y en la segunda iteración 7 estudiantes. Se categorizan los resultados de cada grupo de ejercicio en 3 columnas que hacen referencia a cuál fue la estrategia predominante del alumno para resolver un grupo de ejercicios en particular:

- **Sin dificultad / ayuda mínima:** El estudiante resolvió la mayoría de ejercicios de ese grupo sin dificultad o con una ayuda mínima de nosotros con nuestro rol de facilitador
- **Apoyo herramienta:** El estudiante resolvió la mayoría de ejercicios de ese grupo gracias a la ayuda de la plataforma, ya sea con los carteles de ayuda, mediante el error y los consejos/sugerencias de la plataforma ante el error o en adhesión con una pequeña ayuda de uno de nosotros en nuestro rol de facilitadores luego de la sugerencia de la herramienta.
- **Ayuda facilitador:** El estudiante no logró resolver la mayoría de los ejercicios del grupo ni de forma autónoma ni con el apoyo de la plataforma, por lo que requirió la intervención directa del facilitador para comprender la consigna y la forma de resolverlos.

Clasificamos a los estudiantes en cada una de las columnas de la siguiente manera: si resolvió la mayoría de los ejercicios de un grupo mediante una de las estrategias mencionadas, queda categorizado en la columna correspondiente a dicha estrategia. Los porcentajes hacen referencia al porcentaje total de estudiantes que fueron categorizados en ese grupo.

Tabla 5.1: Resultados porcentuales de resolución por grupo – Iteración 1

Grupo	**Sin dificultad / ayuda mínima**	**Apoyo herramienta**	**Ayuda facilitador**
G1 – Booleanos	50 %	25 %	25 %
G2 – Comparadores	50 %	25 %	25 %
G3 – Operadores lógicos	37.5 %	25 %	37.5 %
G4 – Circuito corto	0 %	12.5 %	87.5 %
G5 – Formalización	50 %	25 %	25 %

Tabla 5.2: Resultados porcentuales de resolución por grupo – Iteración 2

Grupo	**Sin dificultad / ayuda mínima**	**Apoyo herramienta**	**Ayuda facilitador**
G1 – Booleanos	71,4 %	14,3 %	14.3 %
G2 – Comparadores	71,4 %	14,3 %	14,3 %
G3 – Operadores lógicos	57,2 %	14,3 %	28,5 %
G4 – Circuito corto	28,5 %	43 %	28,5 %
G5 – Formalización	71,4 %	14,3 %	14,3 %

Capítulo 6

Conclusiones

En este capítulo se presentan las conclusiones del proyecto, evaluando los resultados alcanzados en relación con los objetivos planteados, así como las proyecciones de trabajo futuro derivadas del análisis realizado.

6.1. Conclusiones del proyecto

El objetivo principal del proyecto consistió en extender la herramienta didáctica existente permitiendo al estudiante enfrentarse a situaciones que están conectadas con conceptos formales de computación de la temática de booleanos por medio de la resolución de pequeños ejercicios sencillos utilizando un pseudocódigo específico. De acuerdo al marco teórico adoptado, el conocimiento informal que se genera en este tipo de interacción con distintos ejercicios, es una base sólida para etapas de formalización propias del sistema educativo.

Dicho objetivo fue alcanzado ya que se diseñó una secuencia de ejercicios organizada en distintos grupos, donde cada conjunto introduce nuevos niveles de complejidad y se asocia con distintos conceptos básicos. En particular, la comprensión de la ejecución computacional como un proceso secuencial que opera sobre estados de memoria se abordó mediante el trabajo sobre variables, expresiones y operadores booleanos, junto con el circuito corto propio de las condiciones booleanas.

Uno de los aportes centrales del proyecto es la aplicación de la teoría piagetiana al diseño de cada Grupo de ejercicios, a diferencia de propuestas tradicionales, donde los ejercicios se presentan únicamente desde su resolución operativa (Weng y cols., 2010; Jiménez-Hernández y cols., 2020; Sabitzer y cols., 2014). En este trabajo se identifica el rol que cumple cada actividad en la construcción de conocimiento del estudiante, así como se especifica desde cuál o cuáles conceptos se retrotrae dicha actividad.

La progresión diseñada en cada ejercicio busca favorecer el pasaje desde una interacción centrada en el resultado (querer resolver el ejercicio, periferia) hacia una toma de conciencia sobre las acciones (la escritura de las instrucciones y

su ejecución, C) y las propiedades y modificaciones de los objetos (visualizaciones iniciales y resultantes, C′), en línea con la Ley General de la Cognición dando los primeros pasos de hacia la toma de conciencia de aspectos computacionales en línea con la ley extendida. Asimismo, la repetición de situaciones presentando problemas con similitudes y diferencias con los ya resueltos, es una estrategia didáctica basada en el concepto piegetiano de generalización constructiva. Ambos principios se describen en el capítulo marco teórico. Asimismo, cada grupo de ejercicios plantea actividades vinculadas a aspectos críticos de conceptos formales.

El uso del modelo Hybrid Interaction System (HIS) permitió estructurar los ejercicios como ciclos de interacción entre el estudiante y el sistema, donde la retroalimentación visual cumple un rol fundamental en la construcción de significado. En este sentido, la herramienta no solo presenta ejercicios, sino que convierte la ejecución del programa en fuente de información valiosa para el estudiante.

En conjunto, los resultados obtenidos mediante las pruebas permiten afirmar que la herramienta genera un ámbito de intercambio que motiva al estudiante a buscar las soluciones. Muchas de las dificultades encontradas en una primera instancia estuvieron dentro de lo esperado y otras surgieron sin ser esperadas. Todas ellas fueron atendidas y, como se desprende de la Tabla 5.2, los cambios realizados produjeron un mejor desempeño de los estudiantes.

6.2. Trabajo futuro

A partir del desarrollo realizado, se identifican diversas líneas de trabajo futuro tanto desde el punto de vista teórico como práctico.

En primer lugar, sería relevante realizar más pruebas involucrando a docentes de programación ya que si bien Mileva es para uso autónomo de los estudiantes, es indiscutible que la opinión de los docentes resulta fundamental. Esta afirmación surge de la experiencia con el módulo 1 de Mileva, que se encuentra disponible en el portal de UruguayEduca, lo que ha permitido recabar opiniones que en parte motivaron la realización de este proyecto.

Al mismo tiempo y como complemento de lo dicho, una línea natural de evolución consiste en incorporar mecanismos de seguimiento del estudiante, permitiendo a los docentes registrar su progreso, los errores cometidos y las estrategias utilizadas. Esta información podría ser utilizada por los docentes por ejemplo, a la hora de planificar sus clases. Desde el punto de vista del desarrollo de la herramienta sería una ampliación en cuanto a la gestión de actividades y de retroalimentación automática.

La extensión natural de Mileva es, por supuesto, la incorporación de nuevos módulos de contenidos fundamentales en programación, tales como estructuras de control (condicionales e iteraciones), operadores lógicos con las mencionadas estructuras de control y estructuras de datos básicas.

Finalmente, resulta de interés explorar la integración de la herramienta en contextos educativos reales, con el fin de evaluar su impacto en escenarios de

enseñanza formal. Esto permitiría analizar no solo la efectividad de los ejercicios, sino también su articulación con prácticas docentes y currículos existentes.

Referencias

Cazzola, W., y Olivares, D. M. (2015). Gradually learning programming supported by a growable programming language. *IEEE Transactions on Emerging Topics in Computing*. doi: 10.1109/TETC.2015.2446192

da Rosa, S. (2018, October). Piaget and computational thinking. En *Proceedings of cserc 2018: The 7th computer science education research conference.* Saint Petersburg, Russia. (October 10–12, 2018)

da Rosa, S., Cabezas, M., Viera, M., y Gómez, F. (2023). Functional programming and didactics of computational sciences. En *Proceedings of the international conference on informatics in schools.* Springer.

da Rosa, S., y Gómez, F. (2020). A research model in didactics of programming. *CLEI Electronic Journal*, *23*(1), Paper 5.

da Rosa Zipitría, S., y Gómez Frois, F. (2019). Towards a research model in programming didactics. En *Proceedings of the 2019 xlv latin american computing conference (clei)* (pp. 1–8). IEEE. doi: 10.1109/CLEI47609.2019.235064

Gómez, F., y da Rosa, S. (2022). Designing a didactic model for programs and data structures. En *Proceedings of the 51 sadio conference, simposio argentino de educación en informática (saei).*

Grover, S., y Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. En *Proceedings of the 48th acm technical symposium on computer science education (sigcse).* ACM.

Grover, S., Jackiw, N., y Lundh, P. (2019). Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school. *Computer Science Education*. doi: 10.1080/08993408.2019.1568955

Herman, G. L., Loui, M. C., Kaczmarczyk, L., y Zilles, C. (2012). Describing the what and why of students' difficulties in Boolean logic. *ACM Transactions on Computing Education*, *12*(1), Article 3, 28 pages. doi: 10.1145/2133797.2133800

Jiménez-Hernández, E. M., Oktaba, H., Díaz-Barriga, F., y Piattini, M. (2020). Using web-based gamified software to learn Boolean algebra simplification in a blended learning setting. *Computer Applications in Engineering Education*. doi: 10.1002/cae.22335

Rosselli, E., Correa, S., y Guayta, C. (2023). *Modelado didáctico de interacción humano-computadora* (Informe de Proyecto de Grado). Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay. (Supervisores: Sylvia da Rosa y Federico Gómez Frois)

Sabitzer, B., Pasterk, S., y Reçi, E. (2014). Informatics – A child's play?! En *Proceedings of edulearn14 conference*. Barcelona, Spain: IATED.

Salanci, Ľ. (2015). Didactics of programming. *International Journal of Information and Communication Technologies in Education*, *4*(3), 32–39. doi: 10.1515/ijicte-2015-0012

Schulte, C., y Budde, L. (2018). A framework for computing education: Hybrid Interaction System: The need for a bigger picture in computing education. En *Proceedings of the 18th Koli Calling international conference on computing education research* (pp. 1–10). New York, NY, USA: ACM. doi: 10.1145/3279720.3279733

Weng, J.-F., Tseng, S.-S., y Lee, T.-J. (2010). Teaching Boolean logic through game rule tuning. *IEEE Transactions on Learning Technologies*, *3*(4), 319–328.

Anexo A

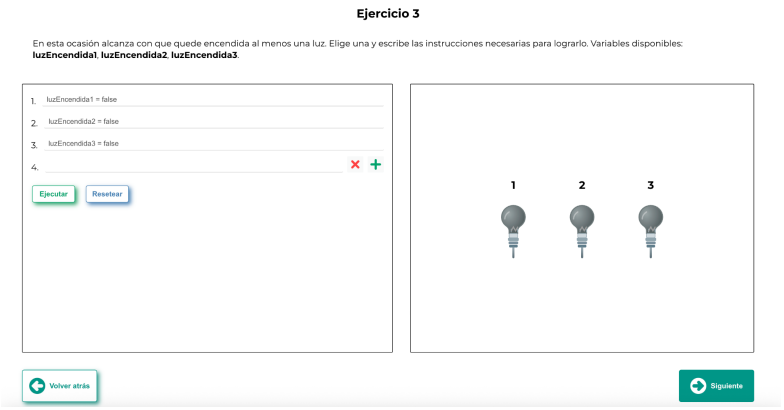
Ejercicios adicionales

En este anexo se incluye el detalle de los ejercicios que, luego de la primera iteración de pruebas, fueron ocultados o movidos fuera del flujo principal de la herramienta. Se preserva su descripción por completitud, ya que pueden servir como referencia para futuras extensiones o variaciones de la secuencia didáctica.

Nota: la numeración que figura en las capturas de pantalla puede no coincidir con la numeracion final utilizada.

Grupo 1

Ejercicio 8



Este ejercicio introduce una variación respecto del ejercicio 2: en lugar de requerir que las tres luces queden encendidas, la condición de éxito es que al menos una lo esté. El propósito didáctico es que el estudiante comience a distinguir entre metas de alcance total y metas de alcance parcial, y que tome conciencia de que existen múltiples soluciones válidas para un mismo problema.

Esta multiplicidad de soluciones refuerza el proceso de generalización constructiva descrito en el marco teórico (Capítulo 2): el estudiante no memoriza una secuencia fija, sino que comienza a comprender la regla subyacente.

Enunciado y Desafío: El sistema presenta tres bombitas apagadas. La consigna indica: "En esta ocasión alcanza con que quede encendida al menos una luz. Elige una y escribe las instrucciones necesarias para lograrlo."

Estado Inicial del Código: El panel de instrucciones muestra tres sentencias pre-cargadas que no pueden modificarse:

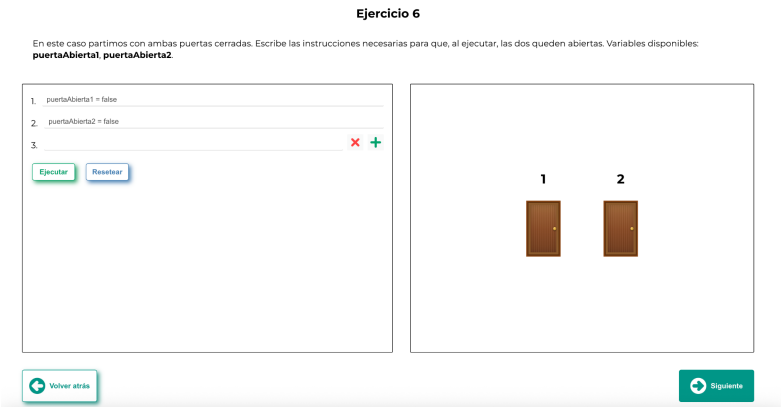
1. luzEncendida1 = false (La bombita 1 está apagada).
2. luzEncendida2 = false (La bombita 2 está apagada).
3. luzEncendida3 = false (La bombita 3 está apagada).

Resolución: El estudiante debe agregar al menos una nueva línea de código que asigne el valor true a cualquiera de las tres variables (luzEncendida1, luzEncendida2 o luzEncendida3).

El sistema valida el éxito con independencia de cuál bombita se elija, siempre que alguna quede encendida al finalizar la ejecución.

A diferencia del ejercicio 2, donde el objetivo era unívoco, aquí el estudiante enfrenta por primera vez la posibilidad de elección. Esto estimula la reflexión sobre la relación entre acción y efecto: se busca que el estudiante comprenda que el cambio de estado no depende de una instrucción "correcta"única, sino de cualquier instrucción que produzca el efecto deseado sobre al menos uno de los objetos del sistema.

Ejercicio 9



Este ejercicio introduce una variación respecto del ejercicio 4: ambas puertas parten cerradas, eliminando el estado inicial ventajoso

en el que una de las puertas ya se encontraba abierta. El objetivo sigue siendo el mismo (ambas puertas abiertas al finalizar la ejecución), pero el punto

de partida exige una intervención completa sobre todas las variables involucradas. Este mecanismo de variación sistemática del estado inicial responde al principio de generalización mediante el contraste, descrito en el marco teórico (Capítulo 2).

Enunciado y Desafío: El sistema presenta dos puertas cerradas. La consigna indica: "En este caso partimos con ambas puertas cerradas. Escribe las instrucciones necesarias para que, al ejecutar, las dos queden abiertas."

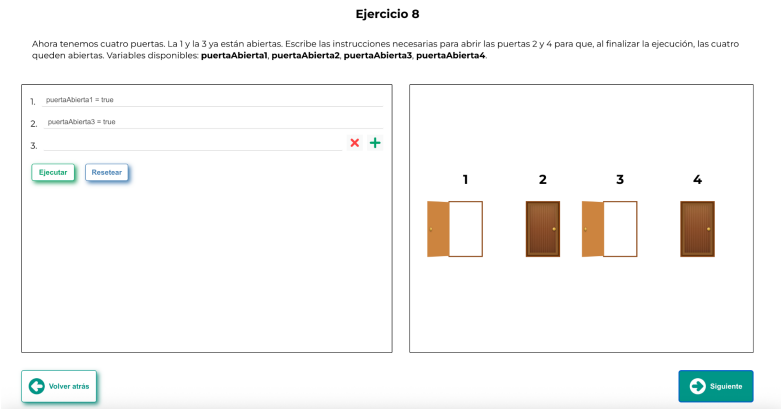
Estado Inicial del Código: El panel de instrucciones muestra dos sentencias pre-cargadas:

1. puertaAbierta1 = false (La puerta 1 está cerrada).
2. puertaAbierta2 = false (La puerta 2 está cerrada).

Resolución: El estudiante debe agregar dos nuevas líneas de código que asignen el valor true a las variables puertaAbierta1 y puertaAbierta2, respectivamente.

Este ejercicio consolida lo aprendido en el ejercicio 5 en un escenario más exigente. Al no haber ninguna variable ya en su estado objetivo, el estudiante no puede limitarse a intervenir sobre una sola puerta: debe operar sobre la totalidad del sistema. Esto favorece la toma de conciencia de que el estado final es el resultado de la composición de todas las instrucciones ejecutadas, y no de una acción aislada.

Ejercicio 10



Este ejercicio eleva la complejidad del escenario de puertas al incorporar cuatro variables booleanas simultáneas, duplicando la cantidad respecto de los ejercicios anteriores del mismo contexto. Además, introduce explícitamente la noción de estado inicial parcialmente favorable: dos puertas ya están abiertas (representadas por instrucciones pre-cargadas no modificables), y el estudiante debe completar el estado del sistema interviniendo únicamente sobre las dos

variables restantes. Esto anticipa la estructura de los ejercicios con alarma que siguen a continuación, donde la comprensión del estado inicial resulta fundamental.

Enunciado y Desafío: El sistema presenta cuatro puertas, de las cuales la 1 y la 3 se muestran abiertas. La consigna indica: "Ahora tenemos cuatro puertas. La 1 y la 3 ya están abiertas. Escribe las instrucciones necesarias para abrir las puertas 2 y 4 para que, al finalizar la ejecución, las cuatro queden abiertas."

Estado Inicial del Código: El panel muestra dos sentencias pre-cargadas que no pueden modificarse:

1. puertaAbierta1 = true (La puerta 1 ya está abierta).
2. puertaAbierta3 = true (La puerta 3 ya está abierta).

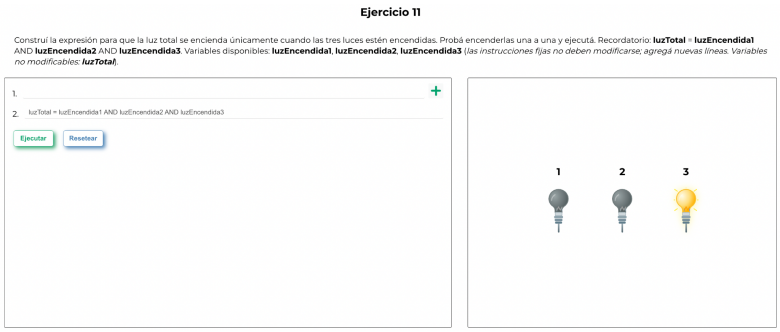
Resolución: El estudiante debe agregar dos nuevas líneas de código que asignen el valor true a las variables puertaAbierta2 y puertaAbierta4.

Este es el primer ejercicio del grupo en el que se trabaja con cuatro variables booleanas de forma simultánea.

Al igual que en el ejercicio 7, el estudiante debe realizar una intervención múltiple. Sin embargo, el salto de tres a cuatro variables introduce una carga cognitiva adicional que refuerza la comprensión de la asignación secuencial como mecanismo general, aplicable a cualquier cantidad de objetos del sistema.

Grupo 3

Ejercicio 12



• **Enunciado y Desafío:** El sistema presenta tres bombitas. La consigna indica: *"Construí la expresión para que la luz total se encienda únicamente cuando las tres luces estén encendidas. Probá encenderlas una a una y ejecutá"*.

• **Estado Inicial del Código:** El panel presenta una configuración que invita a la experimentación:

1. (Espacio vacío para la intervención del estudiante).

2. luzTotal = luzEncendida1 AND luzEncendida2 AND luzEncendida3 (Instrucción fija).

• **Resolución:** El estudiante debe agregar instrucciones para asignar true a luzEncendida1, luzEncendida2 y luzEncendida3. Al ejecutar, Mileva resalta la secuencia en verde, permitiendo al sujeto verificar si su deducción sobre los estados ocultos fue correcta al observar el encendido de la luzTotal. En el panel de visualizacion se puede notar que luzEncendida3 se encuentra ya encendida, con lo cual se podrian esperar inicializaciones en true solamente de luzEncendida1 y luzEncendida2 y no de luzEncendida3. Más aún, también podrían ser de esperar inicializaciones para luzEncendida1 y luzEncendida2 del estilo: luzEncendida1 = luzEncendida3 y luzEncendida2 = luzEncendida3.

Este ejercicio representa un hito en la transición hacia el pensamiento formal dentro de **Mileva.** A diferencia de los ejercicios iniciales de luces donde el estado de cada bombita era siempre evidente, este reto introduce la opacidad de los datos, obligando al estudiante a despegarse de la observación directa para confiar en la **lógica deductiva** y la manipulación de la memoria interna del autómata.

Este ejercicio busca reforzar la importancia de la inicialización y la coordinación. El estudiante experimenta que en un sistema con múltiples dependencias, el fallo en una sola celda de memoria (aunque no figure en las instrucciones iniciales) invalida el proceso total.

Ejercicio 13

• **Enunciado y Desafío:** El sistema presenta tres puertas en el panel visual. La consigna indica: *"Solo se puede cruzar si todas las puertas requeridas están abiertas. Abrilas y combiná la expresión para que puedeCruzar quede true en ese caso"*.

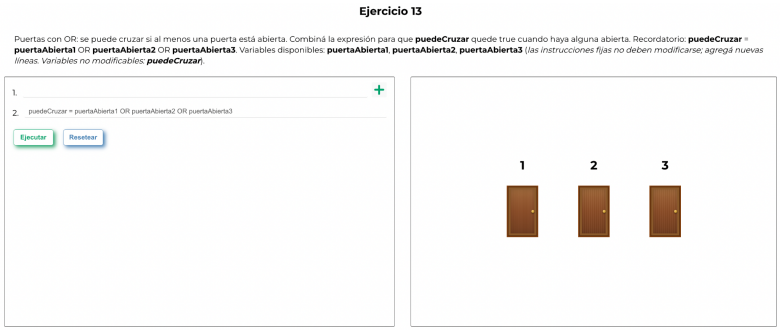
• **Estado Inicial del Código:** El panel muestra una expresión lógica de recordatorio y un espacio para la intervención activa:

1. (Espacio vacío para que el estudiante agregue instrucciones).
2. puedeCruzar = puertaAbierta1 AND puertaAbierta2 AND puertaAbierta3.

• **Resolución:** El estudiante observa que algunas puertas están cerradas en el panel visual. Debe utilizar el botón + para agregar las instrucciones de asignación (ej. puertaAbierta3 = true) que completen los requisitos de la línea 2. Al presionar "Ejecutar", Mileva procesa la conjunción triple y valida si el camino queda habilitado para que el personaje cruce.

Ejercicio 14

Este ejercicio refuerza la gestión de estados múltiples utilizando el operador de conjunción (AND) en un entorno de navegación física simulada. A diferencia de los retos anteriores donde se activaban funciones del robot, aquí el objetivo es permitir el paso a través de un sistema de obstáculos secuenciales, reforzando la idea de que la veracidad de un sistema complejo depende de la integridad de todas sus partes.



• **Enunciado y Desafío:** El sistema presenta tres puertas cerradas en el panel visual. La consigna indica: *"Se puede cruzar si al menos una puerta está abierta. Combiná la expresión para que puedeCruzar quede true cuando haya alguna abierta"*.

• **Estado Inicial del Código:** El panel muestra un recordatorio de la regla lógica y un espacio para la intervención activa:

1. (Espacio vacío para que el estudiante agregue instrucciones de apertura).
2. puedeCruzar = puertaAbierta1 OR puertaAbierta2 OR puertaAbierta3.

• **Resolución:** El estudiante observa que las tres puertas están inicialmente cerradas en el panel visual. Debe utilizar el botón + para agregar al menos una instrucción de asignación (por ejemplo, puertaAbierta2 = true) para satisfacer la expresión de la línea 2. Al ejecutar, Mileva procesa la disyunción triple y valida si el camino queda habilitado para que el personaje avance.

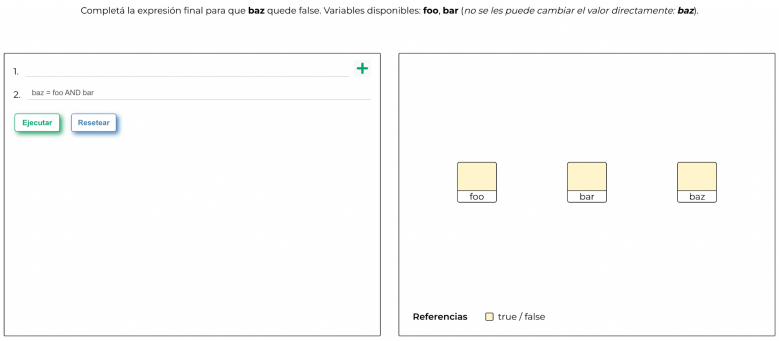
Este ejercicio introduce al estudiante en la gestión de estados múltiples utilizando el operador de disyunción (**OR**) en un entorno de navegación física. A diferencia del ejercicio anterior, donde el paso estaba restringido por la necesidad de abrir todas las puertas (conjunción), aquí el sistema de control se vuelve

más flexible, permitiendo el avance mediante la suficiencia lógica de cualquiera de las rutas disponibles.

Con ello se busca que el estudiante diferencie por contraste la noción de "necesidad conjunta"(producida por el AND) de la noción de "posibilidad múltiple"propia del operador OR. Se busca que el sujeto comprenda que el estado global de éxito (puedeCruzar) puede alcanzarse a través de diferentes caminos independientes entre sí.

Grupo 5

Ejercicio 8



Este ejercicio plantea una experiencia sobre el problema clásico del intercambio de valores entre dos variables (x , w) utilizando una tercera variable auxiliar (z), pero despojado de cualquier analogía física como etiquetas o recipientes, conectando con el concepto formal de sentencia de asignación. A diferencia del módulo 1 de Mileva, se usan aquí variables booleanas.

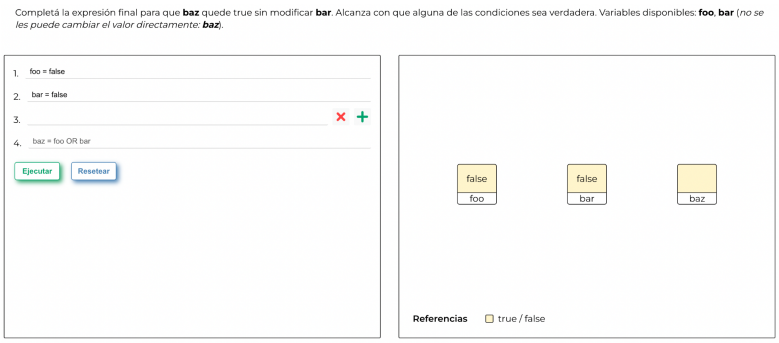
• **Enunciado y Desafío:** El sistema presenta tres cajas abstractas (x, w, z). El objetivo es intercambiar los contenidos de x y w. Para evitar soluciones basadas en la observación directa (hardcoding), los valores iniciales son desconocidos y cambian en cada ejecución.

• **Estado Inicial del Código:** El panel de código está vacío, otorgando al estudiante total autonomía para proponer la lógica de resolución.

• **Resolución:** El estudiante debe diseñar una secuencia de tres asignaciones:

1. z = x (Resguardar el valor de x en la variable auxiliar).
2. x = w (Sobrescribir x con el valor de w).
3. w = z (Recuperar el valor original de x desde la auxiliar y asignarlo a w). Al ejecutar, Mileva muestra mediante animaciones cómo los valores se copian de una celda de memoria a otra.

Ejercicio 9



Este ejercicio representa un avance en la autonomía del estudiante dentro del bloque de formalización al introducir variables de entrada editables en un contexto abstracto. A diferencia de los retos anteriores donde las condiciones iniciales eran fijas, aquí el sujeto debe configurar el entorno de datos para que una expresión lógica predefinida de disyunción (**OR**) resulte verdadera.

- **Enunciado y Desafío:** El sistema presenta tres cajas abstractas (*foo, bar, baz*). Las variables *foo* y *bar* están inicializadas en false pero son editables. La consigna indica: *"Completá la expresión final para que baz quede true sin modificar bar. Alcanza con que alguna de las condiciones sea verdadera"*.
- **Estado Inicial del Código:** El panel presenta una configuración de entrada y un espacio para la resolución:

1. foo = false (Instrucción editable por el estudiante).
2. bar = false (Instrucción editable pero restringida por consigna).
3. (Espacio vacío para la intervención activa).
4. baz = foo OR bar (Instrucción de evaluación fija).

- **Resolución:** El estudiante debe modificar la primera línea para asignar true a *foo*. Al ejecutar, Mileva resalta la secuencia en verde. El estudiante observa cómo el autómata lee el nuevo valor de la celda de memoria *foo* para validar la disyunción y transformar el estado visual del cuadrado *baz*.

Anexo B

Arquitectura

B.0.1. Lenguaje de sentencias de la versión original

Esta sección corresponde al contenido de la sección 5.1.2 del documento de referencia (Rosselli y cols., 2023), que describe el lenguaje simplificado sobre el que se construyó la primera versión de Mileva y que constituye la base sobre la cual la extensión de 2026 amplía el dominio.

Descripción del lenguaje original

Se definió un lenguaje sencillo para las sentencias que pueden ser escritas por los estudiantes. Se optó por simplificar los aspectos de sintaxis —por ejemplo, no se exige el uso de punto y coma luego de una sentencia— para que los estudiantes puedan concentrarse en la semántica asociada a cada instrucción.

Se decidió que el lenguaje sea **fuertemente tipado**: el tipo de una variable es una característica fundamental de la misma y es necesario construir este conocimiento en una etapa inicial del aprendizaje. En el lenguaje definido por la herramienta, el tipo inicial de una variable no cambia, y una variable no puede ser asignada a otra de tipo distinto. El tipo de cada variable es dado por el contexto del ejercicio: en los primeros dos conjuntos todas las variables son numéricas; en el tercer conjunto todas son de tipo *string*; y en el último se tienen variables de ambos tipos.

Tipos de sentencias válidas

Las sentencias válidas permitidas por la herramienta en su versión original son las tres formas presentadas en el Cuadro B.1.

El Tipo 1 (`variable = valor`) es la forma más simple de asignación, equivalente a inicializar una posición en memoria con un valor conocido.

El Tipo 2 (`variable1 = variable2`) permite que el estudiante construya conocimiento sobre la *doble semántica* de las variables: cuando el nombre aparece a la izquierda del operador de asignación referencia el *espacio en memoria*; cuando aparece a la derecha, referencia el *valor* almacenado en ese espacio.

Tabla B.1: Tipos de sentencias del lenguaje original de Mileva (Rosselli y cols., 2023, Sección 5.1.2).

Tipo	Forma sintáctica	**Descripción**
Tipo 1	`variable = valor`	Asigna un literal (número o *string*) directamente a la variable.
Tipo 2	`variable1 = variable2`	Copia el valor de `variable2` en `variable1`. Ambas deben ser del mismo tipo.
Tipo 3	`variable1 = variable2 op valor`	Opera aritméticamente (`+`, `-`) sobre `variable2` y asigna el resultado a `variable1`. Solo variables numéricas.

El Tipo 3 (`variable1 = variable2 op valor`) introduce la noción de transformación aritmética. El resultado de la operación entre `variable2` y un valor numérico es almacenado en `variable1`. Este tipo es el que se necesita para resolver los ejercicios de posicionamiento relativo en el primer conjunto.

Extensión del lenguaje en la versión 2026

La extensión conserva íntegramente los tres tipos de sentencias originales e incorpora las nuevas formas del Cuadro B.2 para soportar la evaluación de expresiones booleanas.

Las formas booleanas mantienen el principio de tipado fuerte: una variable booleana solo puede recibir expresiones que evalúen a un valor booleano. Asignar un literal numérico o de texto a una variable booleana produce un mensaje de error específico que guía al estudiante hacia la corrección.

B.0.2. Diagrama de despliegue

El diagrama de la Figura B.1 corresponde a la figura 5.3 del documento de referencia (Rosselli y cols., 2023). La arquitectura de despliegue no fue modificada en la extensión de 2026: Mileva continúa siendo una **Single Page Application (SPA)** ejecutada íntegramente en el cliente, sin servidor de *backend* ni base de datos externa.

La decisión de mantener este modelo de ejecución exclusiva en el cliente responde a los mismos criterios que guiaron el diseño original: accesibilidad sin instalaciones adicionales, posibilidad de uso en entornos con conectividad limitada, y persistencia del progreso del estudiante mediante el *LocalStorage* del navegador.

Tabla B.2: Sentencias booleanas incorporadas en la extensión de 2026.

Tipo	Forma sintáctica	**Descripción**
Bool. literal	`var = true / false`	Asigna un literal booleano a una variable declarada como booleana.
Bool. copia	`var1 = var2`	Copia el valor booleano de `var2` en `var1`.
Bool. AND	`var = expr1 AND expr2`	Conjunción lógica. Resultado `true` solo si ambas subexpresiones son `true`.
Bool. OR	`var = expr1 OR expr2`	Disyunción lógica. Resultado `true` si al menos una subexpresión es `true`.
Bool. NOT	`var = NOT(expr)`	Negación lógica. Invierte el valor booleano de la expresión.
Comparación	`var = expr1 op expr2`	Compara dos expresiones numéricas con `>`, `<`, `>=`, `<=`, `==`, `!=`. El resultado es booleano.

El único cambio relevante en el despliegue durante la extensión de 2026 fue la migración del *bundler*: de *Create React App* a **Vite** (detallado en la sección B.0.5). Este cambio afecta exclusivamente al proceso de desarrollo y generación del artefacto desplegable, sin alterar el modelo de ejecución descentralizado.

B.0.3. Diagrama de componentes: versión original (2023)

La Figura B.2 reproduce la figura 5.4 del documento de referencia (Rosselli y cols., 2023) y muestra la estructura interna de la primera versión de Mileva. Los grupos 2 y 3 se omiten por ser análogos a los representados.

Descripción de los elementos

CodePanel. Componente React reutilizable que actúa como núcleo de la interfaz de programación. Renderiza el panel de código, gestiona la ejecución secuencial de las sentencias y provee *feedback* visual mediante el resaltado de la instrucción activa. Cada conjunto de ejercicios lo instancia pasando como argumento una función *callback* que define la lógica y las validaciones específicas de ese conjunto.

Componentes de grupo (1–4). Cada componente representa un conjunto de ejercicios: orquesta la carga dinámica del ejercicio activo, renderiza el componente de visualización correspondiente e invoca al *CodePanel* con la lógica

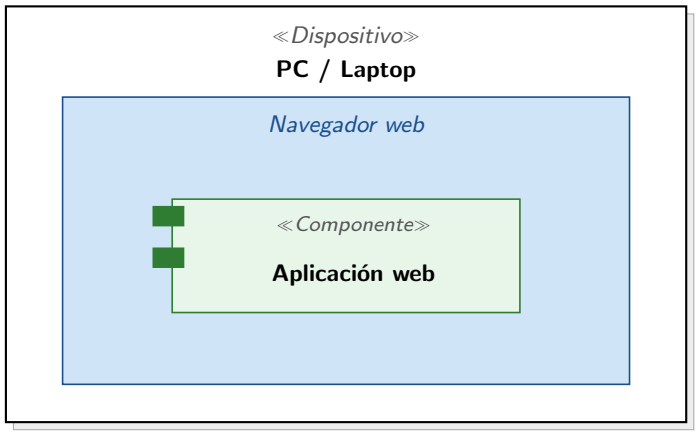


Figura B.1: Diagrama de despliegue de Mileva: un único nodo cliente (PC / Laptop con navegador web). Sin cambios respecto a la versión 2023 (Rosselli y cols., 2023).

del grupo.

Módulo Parser. Módulo JavaScript compartido que expone la función `parseInput(sentence, variables)`. Utiliza expresiones regulares para determinar si la sentencia es sintácticamente válida, realiza chequeos de tipos y devuelve un diccionario con el tipo de sentencia reconocido, o un mensaje de error.

Módulo Arithmetic. Módulo JavaScript compartido que expone la función `calculateNewValue(parsed, variables)`. Determina el nuevo estado de memoria luego de ejecutar una sentencia validada por el *Parser*, sincronizando la lógica del programa con la representación visual.

B.0.4. Diagrama de componentes: versión actualizada (2026)

La Figura B.3 muestra la arquitectura completa de Mileva tras la extensión booleana, diferenciando los grupos del **Módulo 1** y los del **Módulo 2**. El cambio arquitectónico central es la incorporación del módulo `booleanExpressions.js`, que opera como una **cadena de evaluación paralela e independiente** del par *Parser* + *Arithmetic*:

- Los grupos del **Módulo 1** siguen la cadena existente: `parseInput()` → `calculateNewValue()` (*Parser* + *Arithmetic*).
- **Todos los grupos del Módulo 2** (Variables booleanas, **Expresiones booleanas**, Operadores lógicos, Evaluación por circuito corto y Formalización) llaman directamente a `validateBooleanExpressionInput()` de `booleanExpressions.js`, *sin pasar por Parser ni Arithmetic*. Este módulo parsea, valida y evalúa la expresión en un único paso, devolviendo el nuevo estado completo.

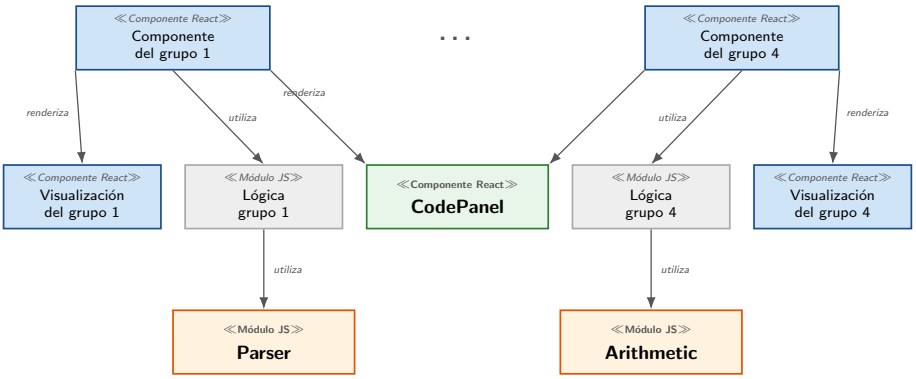


Figura B.2: Diagrama de componentes de la versión original de Mileva. Cuatro grupos de ejercicios, *CodePanel* compartido, módulos *Parser* y *Arithmetic* (Rosselli y cols., 2023, fig. 5.4).

El módulo `booleanExpressions.js` y su relación con el Parser

Motivación. Los primeros cuatro grupos del Módulo 1 y el Grupo 1 del Módulo 2 (Variables booleanas) trabajan con sentencias estructuralmente simples: asignaciones de la forma `var = valor`, copias entre variables y operaciones aritméticas de la forma `var = var op valor`. Estas formas son reconocidas por el módulo `parser.js` mediante expresiones regulares, y la evaluación del nuevo valor se delega a `arithmetic.js`. En el Grupo 1 del Módulo 2 se agregó soporte para la forma `var = varA AND varB` entre variables booleanas, que también fue incorporada en `parser.js`.

A partir del **Grupo 2 (Expresiones booleanas)**, los ejercicios requieren que el estudiante construya expresiones lógicas arbitrariamente compuestas: combinaciones de `AND`, `OR`, `NOT`, comparadores relacionales (`>`, `<`, `>=`, `<=`, `==`, `!=`) y paréntesis. Extender las expresiones regulares del *Parser* original para cubrir este espacio hubiera implicado una explosión combinatoria de patrones y lógica de precedencia difícil de mantener. Por eso se decidió introducir un **módulo paralelo e independiente**: `booleanExpressions.js`.

Funcionamiento de `booleanExpressions.js`. La función exportada `validateBooleanExpressionInput(input, state, options)` implementa en un solo módulo lo que el par Parser + Arithmetic hace para las expresiones numéricas:

1. **Extracción de LHS y expresión.** Con una expresión regular `lhs\s*=\s*expr` separa el nombre de la variable receptora (`lhs`) del cuerpo de la expresión (`expr`).

2. **Validaciones didácticas (opcionales).** Mediante el parámetro `options`,

cada grupo puede configurar restricciones específicas sin modificar el módulo:

- `exactBinaryForLhs`: obliga a que la expresión de una variable tenga exactamente la forma `v1 OP v2`.
- `exactUnaryNotForLhs`: obliga a que la expresión sea `NOT var`.
- `requireVarsInExprForLhs`: exige que ciertas variables aparezcan en la expresión.
- `disallowBooleanOperators`: para grupos de Expresiones booleanas que solo deben usar comparadores numéricos.
- `disallowDirectBooleanLiteralFor`: impide asignar `true`/`false` directamente a ciertas variables, forzando al estudiante a construir la expresión con otras variables.

3. **Normalización y evaluación.** Los operadores `AND`/`OR`/`NOT` se traducen a sus equivalentes JavaScript (`&&`/`||`/`!`), los nombres de variables se reemplazan por accesos al objeto de estado (`s["var"]`), y la expresión resultante se evalúa dinámicamente con `new Function()` dentro de la función `buildExpression`.

4. **Devolución del nuevo estado.** A diferencia de `parseInput()` que devuelve un descriptor estructural {`type, groups`}, este módulo devuelve directamente {`status, data`}, donde `data` es el estado completo actualizado con el nuevo valor de `lhs`. No se requiere llamar a `calculateNewValue()`.

Comparación entre versiones

Tabla B.3: Comparación entre la arquitectura de la versión original (2023) y la arquitectura extendida (2026).

Elemento	**Módulo 1 (2023)**	**Módulo 2 (2026)**
Grupos de ejercicios	4: asignación, orden, intercambio, formalización	5: vars. bool., **expr. bool.**, oper. lógicos, circ. corto, formalización
Parser	3 tipos de sentencia (numérico / *string*)	+ sentencias booleanas (AND/OR/NOT/comparadores)
Arithmetic	Suma y resta sobre valores numéricos	+ evaluación booleana y circuito corto
Tipo de variables	Numérico y *string*	+ booleano (`true`/`false`)
Datos de ejercicios	Definidos *inline* en el componente	JSON en `src/data/exercises/`
Bundler	Create React App	Vite (ver B.0.5)

B.0.5. Migración a Vite y optimización de la infraestructura

Contexto: Create React App

La primera versión de Mileva fue construida con **Create React App (CRA)**, la herramienta de *scaffolding* oficial de React en 2023. CRA provee una configuración de *webpack* preconfigurada que abstrae completamente el proceso de construcción, permitiendo iniciar el desarrollo sin configuración.

Sin embargo, CRA presenta varias limitaciones para un proyecto en evolución activa:

- **Tiempos de inicio lentos:** *Webpack* debe empaquetar la totalidad del código antes de servir la primera página; en proyectos medianos esto toma entre 10 y 30 segundos.
- ***Hot Module Replacement* (HMR) lento:** Cada modificación fuerza la reconstrucción del *bundle* completo; el tiempo de refresco puede ser de varios segundos.
- **Sin mantenimiento activo:** Create React App fue oficialmente discontinuado por el equipo de React en 2023, sin actualizaciones de seguridad.
- **Dependencias desactualizadas:** El ecosistema de CRA arrastra versiones antiguas de Babel, *webpack* y ESLint que generan conflictos con paquetes modernos.

Solución: Vite

Vite es una herramienta de construcción de nueva generación desarrollada por Evan You (creador de Vue.js). A diferencia de *webpack*, aprovecha los **ES Modules nativos del navegador**: en lugar de empaquetar todo el código antes de servir, transforma cada módulo individualmente con *esbuild* (escrito en Go) bajo demanda. Esto hace que el servidor de desarrollo arranque en menos de un segundo.

Principales ventajas:

- **Arranque instantáneo:** típicamente bajo 500 ms.
- **HMR ultrarrápido:** solo el módulo modificado se reenvía.
- ***Build* de producción optimizado con Rollup:** *bundles* comprimidos con *tree-shaking* agresivo.
- **Configuración mínima:** `vite.config.js` es notablemente más simple que la configuración equivalente de *webpack*.
- **Soporte nativo** para TypeScript, JSX y CSS Modules.

Proceso de migración

1. **Eliminación de dependencias CRA.** Se removió `react-scripts` y sus dependencias transitivas de `package.json`.
2. **Instalación de Vite y *plugin* React.** Se instalaron `vite` y `@vitejs/plugin-react` como dependencias de desarrollo.
3. **Creación de `vite.config.js`.** Configuración mínima con el *plugin* React habilitado y las rutas de alias necesarias.
4. **Migración del punto de entrada.** `public/index.html` fue movido a la raíz del proyecto y actualizado con `type="module"` en el *script*.
5. **Actualización de variables de entorno.** El prefijo `REACT_APP_` fue renombrado a `VITE_`; `process.env` fue reemplazado por `import.meta.env`.
6. **Actualización de *scripts* npm.** `start` → `vite dev`; `build` → `vite build`.

Impacto en la arquitectura

La migración a Vite es una modificación **exclusivamente de la capa de herramientas de desarrollo** y no altera ningún aspecto del comportamiento en *runtime*:

- El modelo de ejecución SPA en el cliente permanece idéntico.
- La estructura de componentes React, el *Parser* y el *Arithmetic* no fueron modificados.
- El *LocalStorage* continúa siendo el único mecanismo de persistencia.
- La URL de acceso y el comportamiento observable por el estudiante son exactamente los mismos.

Los únicos cambios visibles para el desarrollador son: tiempos de arranque y recarga significativamente menores, y una estructura de proyecto ligeramente diferente (`index.html` en la raíz, uso de `import.meta.env`).

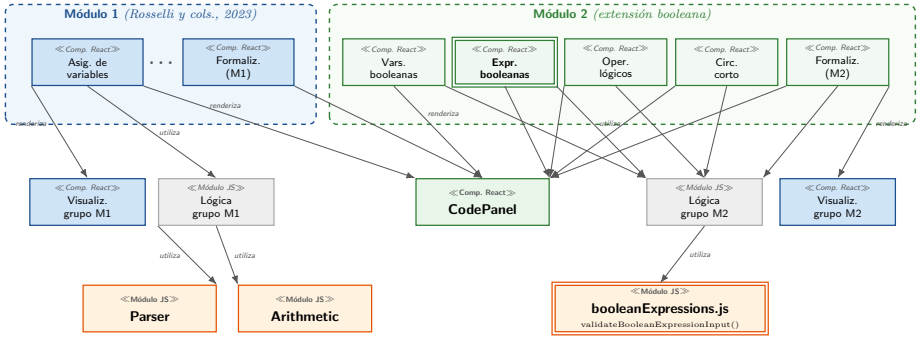


Figura B.3: Diagrama de componentes de Mileva (versión completa, 2026). Los grupos del **Módulo 1** utilizan la cadena *Parser* + *Arithmetic*. Todos los grupos del **Módulo 2** utilizan `booleanExpressions.js`, que parsea, valida y evalúa la expresión booleana en un único paso.